NATURAL LANGUAGE PROCESSING

Classification of News Articles by News Topic

By Shaelin Naidoo

Student Number (218003390)

Table Of Contents

INTRODUCTION

Area Of Application:

This project falls into the area of Text classification. It uses supervised machine learning techniques to model the way articles from different pages of the news differ from each other.

Objective:

To build a classification model that takes in a news article and returns the category the article belongs to.

Dataset:

The source dataset consisted of 2225 BBC news corresponding to stories in five topical areas from 2004-2005. Articles fall into one and only one of the following categories:

- Politics
- Sport
- Tech
- Entertainment
- Business

The dataset took the form of several separate .txt files, each containing a full article, and residing within a folder named for the category of news

DESIGN CHOICES AND OBSTACLES

Formal Language model

The input dataset was not suitable to be used on the classifier. During preprocessing, there are functions that act as analogs to finite state transducers. Since each transducer either deals with a binary problem (retain input or replace input with ε) or is simply implemented as a call to an existing API function, they have been written as simple if else problems or as said function calls respectively instead of through a matrix.

The first and most simple of the transducers applied is the python str.lower() function, which simply converts uppercase letter characters to their lowercase form.

Another one of the transducers uses unicode "**words**" as its input and output language. Its function is to iterate through a string of unicode characters on a word by word basis, as defined by NLTK's word tokenizer. If the word is a member of the english stopword set, it is ignored, i.e replaced with ε. Else it makes it into the output string as is.

The next transducer works on an alphabet of unicode **characters**. It iterates through a string and retains only spaces and characters which are letters, i.e not numbers or punctuation. This is done through the python str.isalpha() function

There are also two NLTK models that are also given access to the string documents. They are the Snowball Stemmer and Wordnet Lemmatizer.

NLP Techniques:

- Tokenization
    - Sentence tokenization was employed to split whole articles into smaller pieces, which each were then considered documents in their own right. This decreases the average length of a document. It was implemented so the model is trained one the actual sentences, allowing the final classify method to function by returning the most frequent prediction of all the sentences within itself.
    - Word tokenization was employed to break down each document when attempting to enhance the distinctive characteristics of the document, i.e, to extract only the most useful words by locating and eliminating stop words, numerics and punctuation.
    - Ngram tokenization is used in the second stage of the experiment to determine if the optimal ngram range for the classification model

- Stemming and Lemmatization
    - Used to reduce words that have undergone morphological transformations to their stem/lemma. This allows words inflected/derived from the same root to be treated the same as each other and their root when fitted to the machine learning model. As a side-benefit, they reduce the overall number of features and make the model less resource-intensive. While they accomplish a similar task, both methods have been employed so as to make a comparison between their efficacy in this application.
- POS tagging
    - In order to perform lemmatization, documents needed to undergo part-of-speech tagging

- Stopword removal
    - Used to filter out clutter in documents, since stopwords take up a lot of space and carry very little information due to their ubiquity. This eases the resource usage and speeds up calculations.

- Vectorization (Count and TF-IDF)
    - SKLearn machine learning models require numeric inputs, so a vectorization step was added to the pipeline to convert text into a vector representation. Both count vectorizer and tf idf vectorizer were used to compare their effectiveness against each other

Programming Language

Due to the powerful APIs available and the ease of use of the language, the language chosen was python (python 3). To make use of Google colab, the code was made compatible with python v3.6.9.

Backups to Secondary Storage

A choice was made to store the data on the local hard drive at each stage up until they were ready to be fitted to the model. This trades the extra storage space needed in exchange for the speed increase of not having to run the entire preprocessing system before fitting to the model. It also provides security in case a stage in preprocessing crashed or was cut short by power outages or other failures.

Classifier Algorithms

The model set-up was streamlined using a Scikit pipeline.
Since preprocessing was done externally, the pipeline contained two stages: a Vectoriser (count or tfidf) and a Classifier from the list below:

- SVC
- Random Forest
- KNN
- Complement Naive Bayes
- Multinomial Naive Bayes
- Logistic Regression (One vs Rest)
- Logistic Regression (Multinomial)
- Multi-Layer perceptron

The only non default parameter was the max_iter value for logistic regression.
The value was set to 500 whenever the default cap on iterations (100) was hit before convergence.

Unigrams only (at first)

Since many classifiers were to be tried, and some were expected to take large amounts of time to train, even on unigrams, it was decided against using higher level n grams in the first round of tests of the experiment. Once the three best models were identified based on unigrams, they were retested on higher level n grams to determine their impact without having to double or triple the total number of trials.

Shortcomings and Obstacles

1) Unsuitable dataset

The original project was intended to work on full length novels and classify them based on their genres. Novels can fall into multiple genres, which would have required multi-label classification.

Finding a suitable dataset for this was challenging since it would take a massive corpus in order to provide a good general picture of the differences between novel genres.

Novels are less information-dense than articles when it comes to features that can help with discerning their genres, as many novels with different genres can employ similar language, and the vocabulary of different chapters within a single novel may also vary based on the themes in the chapter (e.g a romantic scene in a high-fantasy novel).

Since the experiment was done in the free tier of Google Colab, there was a ceiling of 13GB for RAM. The data would have to come from a smaller corpus to not crash the runtime.

It was decided that news articles are a more suitable set of literature to build a classifier on

2) Hand coded vectorizer vs existing vectorizer

Originally, there was to be a custom built vectorizer. This was forgone in favour of SKLearn's existing vectorizers since the provided vectorizers were much faster. The remnants of this are the unused .py files called SCRIPT_3_define_vocab.py and SCRIPT_4_encode.py.

3) Lemmatizing after stopwords

Although the main experiment model performs well enough on the "lemmatized" model, a discrepancy in the PREPROCESSING module meant that the text was not fully lemmatized. This is because stopword removal was performed before lemmatization, which disrupts the POS tagging that is needed for proper lemmatization. Stemmed models did not have a similar issue. This is discussed further under Experiment, Results and Code Walkthrough.

EXPERIMENT

Variables

Independent Variables:

- Classifier type (listed above in design choices)
- Vectorizer type (count or tfidf)
- Train fraction (0.6, 0.7, 0.8)
- Stemmed or lemmatized

Measured variables:

- Accuracy (test accuracy)
- Confusion matrices

Process

- PREPROCESSING

1) Run the .py files:
   SCRIPT_0_adapt_new_dataset.py
   SCRIPT_1_simplify.py
   SCRIPT_2_split_train_test.py
   In that order on a local machine. This handles all preprocessing as well as the train test splits.
   - A choice was made to produce analogous datasets for both stemmed and lemmatized versions, as well as to produce train and test datasets for three different train:test ratios : 7:3, 6:4 and 8:2
   - Scripts 3 and 4 as well as the encoded train test datasets they produce were not used in the final experiment. The scikit models are to be fitted with the textual datasets. However, the vocab file they produce serves one minor purpose: it provides a count of the total number of features which can be used to calculate a suitable k value for KNN classifier.

- FIRST STAGE TRIALS

2) Make the resultant datasets accessible to the ipynb notebook on Google Colab and perform one train-test round for each unique combination of variables. The vectorizer should use only unigrams, the default ngram_range.
   - Note that some classifiers take orders of magnitude longer to train than others. This could have been alleviated by limiting the number of features considered by the model, although this has not been implemented due to the already small size of the dataset.
3) Record the accuracy rate for each combination in the results excel file. Store the confusion matrix heatmap as an image with the name corresponding to the test number.

- SECOND STAGE TRIALS

4) Identify which three models from these (96) trials are the best and retest them with higher level ngrams to see which range produces the best result. Consider the following ngram ranges:
    a) 1-1 (repeat test although already captured)
    b) 1-2
    c) 2-2
    d) 1-3
    e) 2-3
5) Identify the model that has the highest accuracy

- AMENDMENT:

6) As mentioned in Shortcomings and Obstacles, the lemmatization process was seriously flawed. To compare the fixed lemmatization process to the broken original, retest the highest ranked lemmatized model with the fixed lemmatized dataset. Note that due to the way train test splits are performed this will not be the same training sentences being fitted to the model, so tiny deviations in accuracy most likely do not imply an improvement/ degradation in the model.

Note: To maintain consistency in the form and structure of the data used, the accuracy rating is determined by how well the model can classify an individual sentence. This is because the model was trained on individual sentences. However, the final classify function works by passing an entire article to it, tokenizing by sentence and returning the most frequent prediction.

MAIN RESULTS

The full results data for all tests have been provided in the accompanying RESULTS.xlsx file. The confusion matrices for each test are available as .png files bearing a numeric name that corresponds to the test number in results.xlsx.

Var type	var value	avg accuracy	avg accuracy adjusted
train fraction	0.6	0.788925936	0.826687476
	0.7	0.79435911	0.832357244
	0.8	0.796165579	0.834309497
stem or lemma	stem	0.793016592	0.83098963
	lemma	0.793283825	0.831246515
vectorizer	count	0.759291982	0.835814801
	tfidf	0.827008435	
classifier	MultinomialNB	0.851279127	
	ComplementNB	0.861035768	
	KNN	0.484903748	0.746175244
	Logistic_Regression(Multinomial)	0.849984841	
	Logistic_Regression(OVR)	0.849535561	
	Random_Forest	0.794011392	
	MultiLayer Perceptron	0.808294003	
	SVC	0.846157228	

Figure 1: short summary of results (simple mean accuracy for each variable value)
Since KNN+count vectorizer is an outlier among the results, it distorts the averages on non-classifier variables. The avg accuracy adjusted column contains the averages adjusted to ignore any tests where both countvectorizer and KNN were used together.

First stage:

Highest accuracy

The most accurate test was number 82. The combination of variables was:

- Complement Naive Bayes
- Count vectorizer
- Lemmatization
- 0.8 training fraction

This test produced an accuracy of 86.73%

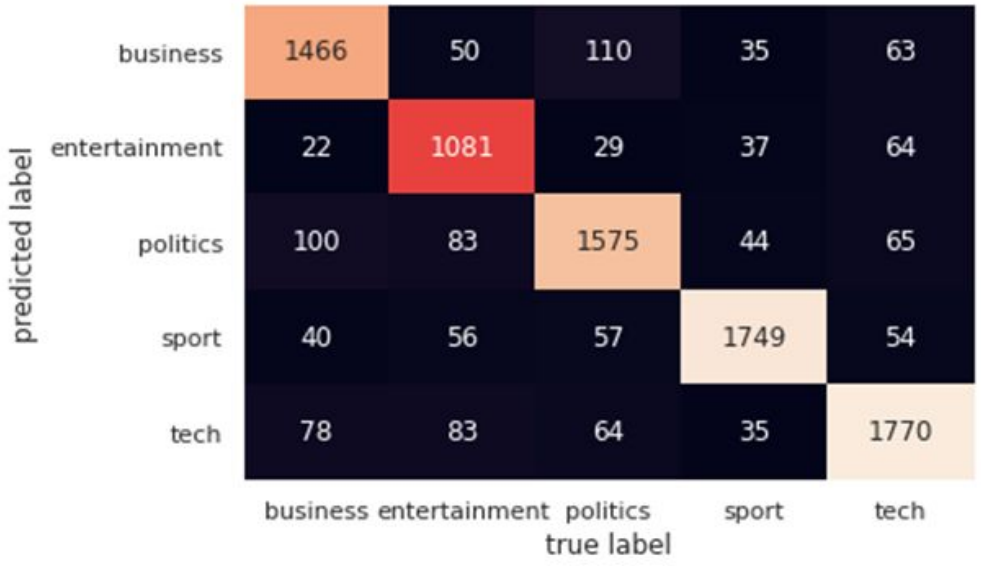

Figure 2: the confusion matrix for test number 82 - the test with the highest accuracy combination of variables in stage 1

The two next best models were tests #96 and #90:

Test #96	**Test #90**
SVC	ComplementNB
TFIDF Vectorizer	TFIDF Vectorizer
Lemmatization	Lemmatization
0.8 training fraction	0.8 training fraction
86.72% test accuracy	**86.31% test accuracy**

Conclusions about vectorizers

The following classifiers favoured tfidf vectorizer

- SVC
- KNeighborsClassifier
- Multilayer perceptron

The following classifiers favoured count vectorizer

- Logistic regression (OVR)
- Multinomial Naive Bayes

The other classifiers have mixed performance per vectoriser based on the rest of the variables

Among models with the tfidf vectorizer, SVC was one of the most accurate classifiers, however, when used with a count vectorizer, their performances became mediocre. Complement Naive Bayes was a strong classifier regardless of the vectoriser.

The least accurate classifier-vectorizer combination was the K Neighbors classifier with the Count vectoriser. K neighbors Classifier performed much better when paired with Tfidf Vectorizer, but this still tended to produce accuracies below what other models were capable of .

Count vectorizer models were slightly quicker to train than tfidf models.

Conclusions about Stemming vs Lemmatization

The difference between the average accuracies of stemmed and lemmatized models is so small that it is negligible. Although the highest 3 ranked models involved lemmatization, so did the single worst model (KNN, countVectorizer, lemmatization,0.6 train fraction) with an accuracy of 22.21%. The stemmed and lemmatised models are distributed evenly across all accuracy ranges.

Stemming a dataset was also performed more quickly than lemmatizing it.

Conclusions about Classifiers

The highest accuracies tend to be produced by the Complement Naive bayes model. This model well outperformed the similar Multinomial Naive Bayes model, which does not account for imbalances in the dataset samples.

Over the top 10 accuracies, complement Naive bayes models took 8 spots, and Support Vector Classifiers took the remaining two.
The K Neighbors Classifier and Random Forest Classifier consistently produced some of the poorest results, followed by the Multi-Layer perceptron classifier

Complement and Multinomial Naive bayes train and test in the order of single digit seconds.
Both logistic regression models train and test within 10 seconds
KNN trains in single digit seconds and tests in 10-20 seconds
Multi-Layer Perceptron, Random Forest and SVC usually take between 10 and 15 minutes.

Without a doubt, the Complement Naive Bayes classifier was the best equipped for the task

note: references to time taken by a model are based on experiences in Google Colab during the experiment. They are bound to vary based on many factors, but provide a useful if rough comparison of how long different classifiers take to train/test relative to each other

Conclusions about Train fraction

From the three train fractions in the experiment, the average accuracies increased as the train fraction increased. 0.8 train to 0.2 test was the optimal ratio in the experiment.

Second stage:

Highest accuracy

The most accurate test was number 99. The combination of variables was:

- Complement Naive Bayes
- Count vectorizer
- Lemmatization
- 0.8 training fraction
- Ngram range 1 to 3

This test produced an accuracy of 88.50%

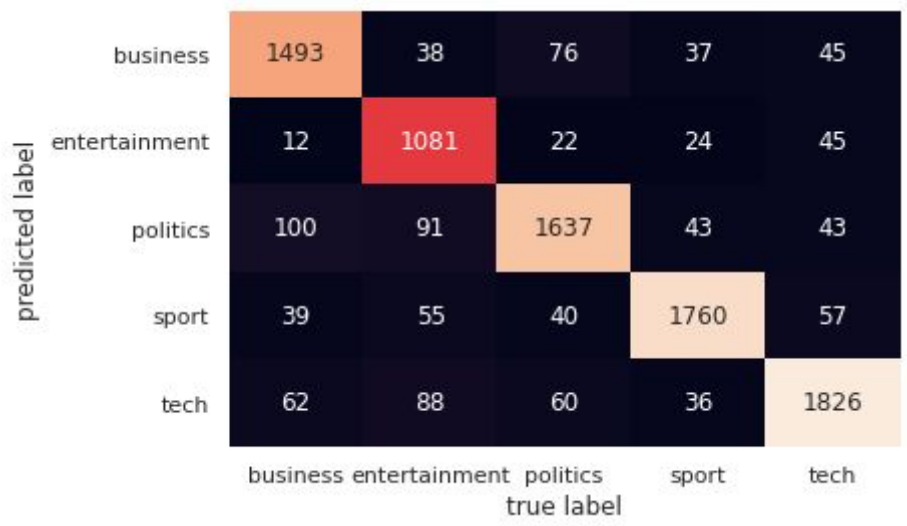

Figure 3: the confusion matrix for test number 99 - the test with the highest accuracy combination of variables in stage 2

Conclusions about ngram ranges

test number	train_fraction	stem_or_lemma	vectorizer	classifier	ngram range	test accuracy
97	0.8	lemma	count	ComplementNB	1,1	0.867309875
98	0.8	lemma	count	ComplementNB	1,2	0.884335982
99	0.8	lemma	count	ComplementNB	1,3	0.885017026
100	0.8	lemma	count	ComplementNB	2,2	0.77014756
101	0.8	lemma	count	ComplementNB	2,3	0.770828604
102	0.8	lemma	tfidf	SVC	1,1	0.867196368
103	0.8	lemma	tfidf	SVC	1,2	0.862996595
104	0.8	lemma	tfidf	SVC	1,3	0.854370034
105	0.8	lemma	tfidf	SVC	2,2	0.728036322
106	0.8	lemma	tfidf	SVC	2,3	0.70261067
107	0.8	lemma	tfidf	ComplementNB	1,1	0.863904654
108	0.8	lemma	tfidf	ComplementNB	1,2	0.878206583
109	0.8	lemma	tfidf	ComplementNB	1,3	0.877866061
110	0.8	lemma	tfidf	ComplementNB	2,2	0.764358683
111	0.8	lemma	tfidf	ComplementNB	2,3	0.765266742

Figure 4: the accuracy rating for all tests in stage 2

Including bigrams saw an increase in both ComplementNB models, but a slight decrease in the SVC model. Using 1,2,and 3 grams saw an increase only in the ComplementNB+Countvectorizer model and not in the other two models.

Excluding unigrams entirely saw a large decrease in accuracy across all models

Corrected Lemmatizer:

The highest ranked lemmatized model happened to also be the highest ranked model overall. The model for this test was thus set up as follows:

- Lemmatized (properly)
- 0.8 train fraction
- CountVectorizer
- ComplementNB

The model was re-trained on the new train dataset which was split independent of the previous train dataset.

During testing it produced an accuracy of 88.46%.
The same test was performed on the second and third highest ranked lemmatized models (also happen to be the 2nd and 3rd ranked models overall). They also experienced a slight drop in accuracy.
This seems to imply that the flawed lemmatizer actually produced a more accurate model, but this cannot be confirmed with so few tests. In fact, in the extra tests, the fixed lemmatizer model had more correct predictions than the flawed one

Final Conclusion on Model:

The ComplementNB Classifier, paired with Countvectorizer produced the best classification pipeline/model. This model should be trained on a dataset that has been lemmatised. 80% of the dataset should be reserved for training.

The ngram range should be either (1,2) or (1,3)

Lemmatization using the flawed lemmatizer does not hinder the model.

EXTRA RESULTS

Trials on data from other sources

After the formal experiment, the model was tested on 10 articles from outside the original dataset. This process was not as controlled as the main experiment and was only performed using one version of the model, the most accurate version.

This time, the classifier was given the entire article and broke it down by sentence, taking the modal prediction of sentences to be the final prediction.

5 of the 10 provided articles were from BBC news (one for each category)

The remaining 5 came from the New York Times, Washington Post, Independent, Guardian and Huffington Post.

Using the original, flawed lemmatizer:

The model was able to correctly identify the genre of a full article in 9/10 tests, with the only misclassification being that of an entertainment article (from the Huffington Post) as a sport article.

Using the fixed lemmatizer:

The model correctly identified all 10 of the test samples

Due to the small sample size it is impossible to draw conclusions about the accuracy of the model on this data

CODE WALKTHROUGH

.py Scripts

The Notebooks are only given access to the final train and test data. All the preparation steps are handled in .py scripts. They are named in the order that they should be called on unprocessed data

SCRIPT_0_adapt_new_dataset.py

- Takes the original, unchanged dataset, which consists of several text files in folders corresponding to category names
- For each text file in a folder, sentence-tokenizes it and adds each sentence to the list of documents for the category (in a dict).
- Finally, saves one text file for each category

SCRIPT_1_simplify.py

- This is the main NLP process.
- It preprocesses the dataset
- lemmatization/stemming and removal of punctuation and numbers is done here.
- saves one text file for lemmatized lines of each category, in the directory DATA/SIMPLIFIED/LEMMATIZED,
- saves one text file for stemmed lines of each category, in the directory DATA/SIMPLIFIED/STEMMED

SCRIPT_2_split_train_test.py

- Produces csv files for training and testing subsets
- Produces sets of these csv files for both stemmed and lemmatized versions

SCRIPT_1_simplify_FIXED.py and SCRIPT_2_split_train_test_FIXED.py

- Serve the collective function of the scripts for which they are named respectively.
- However, **SCRIPT_1_simplify_FIXED** calls the simplify from the fixed preprocessing module, and only produces a lemmatized dataset.
- Likewise **SCRIPT_2_split_train_test_FIXED** only performs the train test split on the lemmatized data, and only performs an 80:20 split since this was the ratio in the most accurate model
- The train and test datasets produced by SCRIPT_2_split_train_test_FIXED will probably not contain the same source sentences as the one produced by **SCRIPT_2_split_train_test**

The following two scripts are not relevant to the scikit models, but were intended to be used to produce a vectorised dataset. However, a choice was made to use the provided vectoriser from Scikit.

SCRIPT_3_define_vocab.py

- Stores the counts of each word in each line in the training data in a dict.
- Then saves this vocab to a csv.
- The saving function from FILE_MANAGEMENT module assigns a unique int id to each word, such that the most frequent word is 1, the second most frequent is 2, etc

SCRIPT_4_encode.py

- Reads csv file, either training or testing subset into a pandas DataFrame. Replaces each word with its id according to the vocabulary. If a word in the testing set is not in the vocabulary it is ignored
- Saves these encoded train and test DataFrames to files

Note: These scripts reference files in a folder structure that existed during the experiment. Deviations from this folder structure will require modifications in the script.
In the submission, only the original data is present, but the folder structure is preserved, so that each stage of preparation can continue unimpeded if the experiment is replicated.

Custom .py Modules

Scripts make reference to these user defined modules

FILE_MANAGEMENT.py

At each stage of preparing data the project backs up the data to files. This module handles the backup and recovery process

SPLIT.py

This module currently contains only one class, TRAIN_TEST_SPLIT, which allows the choosing of a randomized training and testing subset. The main splitting function works by taking a categorised dictionary of article lines, shuffling the data, and picking a random subset of lines for each split. In the experiment it was ensured that the train fraction was enforced not just over the total dataset but within each category itself.

PREPROCESSING.py

Handles the process of enhancing the distinctiveness of a document. It strips numbers and punctuation, and lemmatizes or stems the document. Does not perform vectorization. It is technically flawed when performing lemmatization but has been included since most lemmatization-based model used a dataset based on the broken lemmatizer herein

PREPROCESSING_FIXED.py

Does the exact same thing as the previous item, however it accounts for the fact that lemmatization depends on stopwords. Thus, within the main simplify() function, if the data is to be lemmatized, stopword removal is performed after lemmatization, not before as in the previous module.

LEMMATIZATION.py

The lemmatization process is part of preprocessing, but is clunky and has therefore been encapsulated into its own module. It is only referenced in the method lemmatize() in PREPROCESSING

.ipynb Notebook

SCIKIT_TEXT_CLASSIFICATION.ipynb

This is the notebook with the classifier models, and it operated on already-preprocessed data.

It contains the first and second stage tests as well as the means to get data from the file system into a format readable by the model.

It also contains a demonstration of a simple classify function based on the most accurate model.

Although such a function could be implemented, this classify function does not consider the confusion matrix of the model for tie breaking.

Note: The text segments of the ipynb file provide ample explanation and documentation for the notebook.

REFERENCES

Dataset Source

Hosted by user *Shivam Kushwaha* on Kaggle
https://www.kaggle.com/shivamkushwaha/bbc-full-text-document-classification

D. Greene and P. Cunningham. "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering", Proc. ICML 2006.

APIs used

- Scikit Learn
  https://scikit-learn.org/stable/

- Natural Language Toolkit(NLTK)
  https://www.nltk.org/