

Module Code: CS3TM20

Assignment report Title: Technical report

Student Number: 29015642

Data set information derived from student number:

```
[6] index=input('type your student number?')
type your student number?290156421

My student number is 29015642 but when inputting it gave me a data set index of (2 2) which gave me the same two categories. To prevent this
issue, i added a 1 at the end of my student number and this gave me a different data set index.

[7] x=divmod(int(index),4)
yourdata1=x[1]
y=divmod(int(index),3)
yourdata2=y[1]

print('This is your data set index ----> (', x[1], y[1], ')')

This is your data set index ----> ( 1 0 )

data1= twenty_train.target_names[x[1]]
data2= twenty_train.target_names[y[1]]
categories1=[data1,data2]
print(categories1)

['comp.graphics', 'alt.atheism']
```

Date (when the work completed): 14//03/2023

Actual hrs spent for the assignment: 15h

Assignment evaluation (3 key points):

- 1 Coursework guidelines were a big vague.
- 2 I would have enjoyed the coursework more if we had to code a lot more.
 - 3 NLP was very interesting to learn and apply practically.

Repository including Code and Utils:

<https://csgitlab.reading.ac.uk/jz015642/text-mining-and-natural-language-processing-coursework/-/tree/main/>

Contents

1.0 Abstract.....	3
2.0 Introduction	3
3.0 Methodology.....	4
3.1 NLP	4
3.2 Machine learning	5
4.0 Development, Results, and discussion	6
4.1 The pipeline.....	6
4.2 Test-Train Split	7
4.2 NLP analysis.....	7
4.3 Classifier results	10
4.4 Evaluation and Error analysis.....	11
5.0 Conclusion and future work.....	14
6.0 References and Acknowledgements.....	15

1.0 Abstract

The meaning of words in natural languages are fully reflected by their context and contextual relations. within this research, we explore the use of various Natural Language Processing (NLP) techniques, such as tokenization, stop word removal, and TFIDF, to train machine learning models that can accurately predict the category of a newsletter. We evaluate the performance of our model on a dataset of newsletters from the two different categories, alt.athiesm and comp.graphics, showing that the NLP approach achieves high accuracy. The results demonstrate the effectiveness of combining NLP techniques and machine learning for automatic text classification tasks.

2.0 Introduction

Natural language processing (NLP) is a section of computer science and artificial intelligence that specialises in the algorithms that enable computers to interpret, understand and generate natural language text or speech. The discovery of NLP dates back to the early 1940s [1] shortly after the events of World War II. The importance of NLP was recognised and has the field has since exploded in complexity and understanding. In the modern era, natural language models, such a chatGPT [2], have become so advanced that the capabilities of NLP have no limits. NLP can be applied to almost any text related problem which is why the ability to develop machine learning models based off NLP can be extremely beneficial.

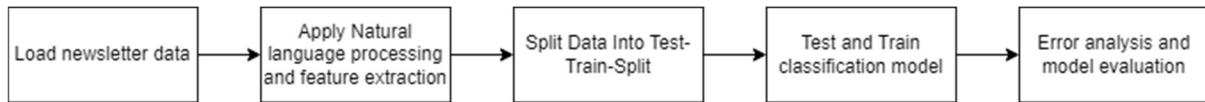
Classifying articles within the newsletter dataset can be a challenging task as it requires accurately allocating each article to its respective category. Hence requiring the need for human inspection which can be very tedious and time consuming. Therefore, the ability to predict the category using machine learning with high accuracy can significantly speed up the process of organizing the articles and allocating labels in a swiftly fashion. Previous works includes the use of a Calibrated Classifier and Linear SVC machine at which multiclass classification was done on the same dataset using One-Hot encoding, frequency vectors and tfidf. The study achieved a weighted model f1 score of 0.88 [3].

Main challenges that exist is the neglection of removal of noise within text. Noise includes various unimportant features in the language that will not support the machine learning model. Tech Target explains how noise can be prominent in unstructured text that is primarily found in blogs, chat conversations, discussions, emails [4]. This study will use a variety of NLP techniques to create an accurate machine learning model that can be used for text classification purposes. The methodology section within this paper explains the theory behind the NLP techniques and machine learning algorithms. Section 4 within this paper explains in detail, the technicalities and related code that will be used. This section also visualises the results and critically compares the performances of each model through each iteration of selected pipeline, giving explanations and discussions in to how to improve the models.

3.0 Methodology

A flowchart revealing the overall design and implementation of the NLP model can be shown in figure A.

Figure A



3.1 NLP

Pre-processing of the text data is one of the most important steps in entire pipeline. Pre-processing converts unstructured data into a structured format so that the data can be analysed by machine learning models more effectively. Text data has a high likelihood of containing a lot of noise. This noise is represented by irrelevant and redundant information that can negatively impact the performance of machine learning models. In the context of text data, pre-processes involve a series of steps that are designed to clean, normalise, and transform raw text data into a format that is much smaller in dimensionality. Making it more suitable for machine learning analysis by the removal of irrelevant information and focusing on only relevant features that are suited for the classification or regression task.

For this research, the following NLP pre-processing and feature extraction techniques have been used:

- Stemming
- Tokenization
- Stop word removal.
- N-grams.
- Term Frequency-Inverse Document Frequency (TFIDF)
- Part of Speech (POS)

Stemming is a type of normalization that involves reducing words to their stems by removing affixes. While stemming can simplify text analysis, it can also lead to the loss of meaning or result in non-words. For instance, applying stemming to the sentence "This was not the map we found in Billy Bones's chest, but an accurate copy" would result in "thi wa not the map we found in billi bone's chest but an accur copi." This shows that stemming can sometimes produce unintelligible text, and its application should be carefully considered depending on the context and purpose of the text analysis.

Tokenization is an approach of breaking text into individual words (or tokens). Tokenisation is a significant pre-processing step such that most NLP algorithms rely on the assumption that the text has been tokenized into a sequence of discrete units. Standardised methods for tokenisation include using deterministic algorithms based on regular expression. A regular expression can be defined as a sequence of characters that define a search pattern within a text or document. Regular expressions can be a beneficial tool for identifying and isolating specific patterns or structures in text data, but tokenization is usually completed through a variety of techniques that may include regular expressions. In general, tokenisation is a critical tool in many NLP tasks that help the preparation of texts for ensuing analysis and feature extraction.

Term Frequency-Inverse Document Frequency (tfidf) is a numerical statistic that reveals the importance of a word in a collection of documents. It can be extremely useful for information retrieval and can be used as a feature into a machine learning model to help classify context of a document. Tfidf can be split into two main components TF and IDF. TF measures the frequency of the word within a document. Where it represents the number of times the word term appears in a document, compared to the number of words in that document. Tf can have issues when overly frequent words, such as "they", "it" and "the" don't give

a lot of information about the context. Fortunately, Tf can be used in conjunction with stopword removal at which these frequent words can be removed before term frequency can be deployed. Idf, on the other hand, looks at how common a word is in all the documents. Idf calculates the commonness of the word by using the number of times the word appears in the document and the entire corpus. A logarithmic operation is applied and the idf is calculated for each term in the document. Idf can therefore be used to help correct words like “the”, “it”, “as”, “of” as they appear much more frequently in written an English corpus [5]. However it is also recommended that stopword removal is used before idf. The weighting of the frequent terms can therefore be minimised, and more infrequent terms can have a more weighted importance. Therefore, when tf and idf are put together, each word in the document has a weighted value which is used to represent how often the word appears across the entire corpus. These techniques can be used to identify words that are of particular importance and can be used as features for machine learning models to help classify different documents/texts based on the importance of each word.

An N-gram is an NLP model that assigns probabilities to sequences of words and sentences. N-grams have combinations of adjacent words or letters that have length n in the text source. N-gram models are attempts to predict the following word in a sentence based on the N-1 previous words in the sentence. N-grams are very important in text mining when the task is to extract patterns from the text.

Stop words are known as the most commonly used words in the English language that convey very little meaning to any context. An example of stop words would include “a” “an” or “the” but there are many more. It is very important in natural language processing that these words are removed. These types of words can take up a lot of memory space and can ultimately increase the time taken for a computer to process the entire text. Therefore, by removing stop words from the text as part of the first pre-processing steps, other complex pre-processing techniques will have significantly less information to process. Overall removing stop words is essential for any NLP task, but especially when it comes to machine learning.

Part of Speech (POS) tagging is a NLP technique that is used to label each word with its corresponding part of speech. For example, assigning a word to a noun, verb, adjective or many other. POS tagging is useful for a variety of NLP tasks, such as information extraction, named entity recognition, and machine translation [6]. Grammatical structure of a sentence can be distinguished using POS tagging which can be very informative. Usually, most POS machine learning algorithms are pretrained on very large, annotated corpora of text so that the algorithms are able accurately assign the part of speech for each word depending on the context of the document.

3.2 Machine learning

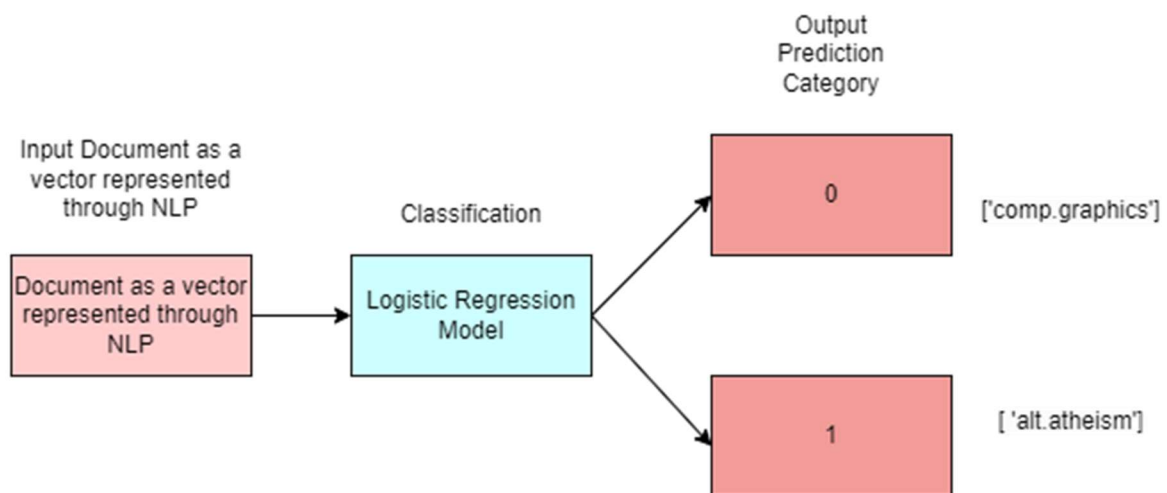
Machine learning is a big part of artificial intelligence such that it involves the training of computer systems to learn information from data and improve performance on tasks. ML has the ability make predictions and decisions based off its learning. Machine learning’s goal is to develop models so good that they can make decisions similar of a human. ML can be applied to any problem and can be extremely useful in speeding up processes that may take a long time. This is especially true in text mining as for a human to read an article to understand the intention or sentiment of the article, it would take the human a while to read the entire article. However, if a machine learning model was trained to automatically determine the sentiment or intention, then this process could take a fraction of the time of a human reading the article.

Logistic regression is a type of supervised machine learning algorithm that can be used for binary or multi-class classification tasks. The goal of a logistic regression model is to predict if something belongs to a certain class based off the set of input features. In our case the logistic regression model would be used to label which category the document belongs to. This type of classification technique learns a linear boundary that separates the classes by estimating probabilities that the features belong to a certain class. A probability threshold is used to distinguish between two or more classes.

In machine learning, the SGD classifier is used for binary and multiclass classification tasks. Using stochastic gradient descent, it optimizes a linear function that separates the training data into different classes. During training, the SGD classifier tries to find the ideal weights for the linear function that minimises a loss function, which measures the difference between the predicted and true labels. The classifier renews the weights in the direction of the negative gradient of the loss function with respect to the weights. The SGD classifier is computationally efficient, adaptable, and can handle non-linearly separable data by using kernel methods. It is widely used in machine learning for binary and multiclass classification tasks due to its efficiency and flexibility [7].

Figure B is used as a representation of one of the chosen machine learning classifiers, Logistic Regression or SGD respectively. The flowchart represents all the inputs and outputs to the machine learning model.

Figure B



4.0 Development, Results, and discussion

The task includes training newsgroup data with a machine learning model with the intention of being able to group the labels of the two categories generated by my student number.

4.1 The pipeline

By using the pipeline function from sklearn [8] we can create a step of instructions for our machine learning model. To begin, our pipeline takes the raw data as its input feature, pre-processes the text using custom NLP functions such as tokenisation and POS, which are fed into a machine learning function CountVectorizer [9]. CountVectorizer is a tool from the sklearn library in python that is used to transform text into a numerical representation. Machine learning models can only take numerical information as its parameters, so this step is necessary for any natural language processing classification task as the raw data is of text (strings). By counting the frequency of each word in the text, a value can be encoded and stored within a matrix. This matrix can then be used as a feature vector for a machine learning model. After the feature vector has been generated, a feature extraction method TFIDF is deployed to apply weights to each value in the feature vector. Now that the data has been prepared, it is fed into the machine learning classifier to be trained. The resulting trained model can then be used to make predictions on new text data.

Code Snippet A shows the pipelines for this research.

Code Snippet A

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.linear_model import SGDClassifier, LogisticRegression

text_clf = Pipeline([
    ('vect', CountVectorizer(binary=True, analyzer=processText)),
    #('vect', CountVectorizer(binary=False, analyzer=processText))
    #('vect', CountVectorizer(binary=True, analyzer=getTags))
    #('vect', CountVectorizer(binary=False, analyzer=getTags)) #preprocessing task
    #('tfidf', TfidfTransformer(use_idf=True)), #feature extraction

    ('clf', SGDClassifier())
    # ('clf', LogisticRegression())
])
```

4.2 Test-Train Split

We split our data into testing and training by allocating a 60/40 training split when directly downloading the news article data.

Code Snippet B

```
twenty_train1 = fetch_20newsgroups(subset='train', categories=categories1, shuffle=True, random_state=42)
twenty_test1 = fetch_20newsgroups(subset='test', categories=categories1, shuffle=True, random_state=42)
```

This has allocated a 708 testing and 1064 training documents. It is always important to have more training data than testing data so the machine learning can learn more patterns and information than it can test on.

It is important to also note that for the training batch 480 out of the 1064 belong to the atheist class whereas 584 belongs to the comp graphics class. This is a sign of the classes being imbalanced and this could be a problem later when we do our machine learning model analysis.

Output A

```
➡ Training set class counts:
alt.atheism: 480
comp.graphics: 584

Test set class counts:
alt.atheism: 319
comp.graphics: 389
```

4.2 NLP analysis

Code snippet C reveals the code used to pre-process the data so that it can be used in the machine learning model. This involves removing information such as hashtags, URLs and miscellaneous characters.

Code Snippet C

```
def processText(text, lemma=False, gram=1, rmStop=True): # default remove stop words
    text = re.sub(r'(https|http)?://(\w|\.|\s|\/|?|\\|&|%)@|@|w+|#', '', text, flags=re.MULTILINE) #delete URL, #hashtag, and @xxx
    tokens = word_tokenize(text)
    whitelist = ["n't", "not", "no"]
    new_tokens = []
    stoplist = stopwords if rmStop else []
    for i in tokens:
        i = i.lower()
        if i.isalpha() and (i not in stoplist or i in whitelist): #i not in ['.', ',', ';'] and (...)
            if lemma: i = lemmaWord(i)
            new_tokens.append(i)
    del tokens
    # tokens = [lemmaWord(i.lower()) if lemma else i.lower() for i in tokens if (i.lower() not in stoplist or i.lower() in whitelist) and i.isalpha()]
    if gram<=1:
        return new_tokens
    else:
        return [' '.join(i) for i in nltk.ngrams(new_tokens, gram)]
```

The text is firstly normalised by removing irrelevant information, such as URLs, hashtags, and other unnecessary characters. This is done by using the regular expression function `re.sub()` which can be used to replace a list of characters. The first parameter is a string of regular expression that contains all the characters that we want to remove. It is worth noting that a “r” string was used to represent these characters. This is because the `r` prefix tells python to treat the string as a raw string and not apply any special processing such as “\n” for new line. The second parameter is what the function will replace those strings/characters with. The third parameter is the input of the raw text data variable, and the final parameter is a flag used to deal with multiline text information.

The `rmStop` parameter is set to `true` such that commonly occurring words such as “the”, “of”, “in” and “is” are removed as they offer very little value in text analysis. A new array is populated from the `stopwordEn` database that contains all the possible stop words in the English language. These values inside the stopwords are then used to compare with the tokens, hence filtering out any stop words in the tokens.

Each token is iterated through the list of tokens and is normalised using `.lower()` to ensure all alphabetical values are lower case as the `stopwordEn` database contains lower case words as well as the whitelist. The tokens are also checked if they contain alphanumeric values. If all these conditions are met, then the token has then been accepted through the NLP techniques and is appended to a new list of tokens called `new_tokens`.

After all the iterations of these NLP techniques, the function will apply ngrams to the tokens if the `gram` value is set to above 1, else the function will return the new set of fully preprocessed tokens without any bigramming.

The newletter on Output B will be used for visualisation purposes to show the effect of each nlp technique.

Output B

```
From: ani@ms.uky.edu (Aniruddha B. Deglurkar)
Subject: help: Splitting a trimming region along a mesh
Organization: University Of Kentucky, Dept. of Math Sciences
Lines: 28

Hi,

I have a problem, I hope some of the 'gurus' can help me solve.

Background of the problem:
I have a rectangular mesh in the uv domain, i.e the mesh is a
mapping of a 3d Bezier patch into 2d. The area in this domain
which is inside a trimming loop had to be rendered. The trimming
loop is a set of 2d Bezier curve segments.
For the sake of notation: the mesh is made up of cells.

My problem is this :
The trimming area has to be split up into individual smaller
cells bounded by the trimming curve segments. If a cell
is wholly inside the area...then it is output as a whole ,
else it is trivially rejected.

Does any body know how this can be done, or is there any algo.
somewhere for doing this.

Any help would be appreciated.

Thanks,
Ani.
--
To get irritated is human, to stay cool, divine.
```

The tokenisation result of the raw text data from Output B is shown in Output C

Output C

```
-----tokenize:
['From', ':', 'ani', '@', 'ms.uky.edu', '(', 'Aniruddha', 'B.', 'Deglurkar', ')', 'Subject', ':', 'help', ':', 'Splitting', 'a', 'trimming', 'region', 'along', 'a', 'mesh',
Amount of tokens created: 216
```

Output D is the result of applying stemming to the raw data from Output B

Output D

```
-----stem:
from: ani@ms.uky.edu (aniruddha b. deglurkar)
subject: help: splitting a trimming region along a mesh
organization: university of kentucky, dept. of math sciences
lines: 28

hi,

i have a problem, i hope some of the 'gurus' can help me solve.

background of the problem:
i have a rectangular mesh in the uv domain, i.e the mesh is a
mapping of a 3d bezier patch into 2d. the area in this domain
which is inside a trimming loop had to be rendered. the trimming
loop is a set of 2d bezier curve segments.
for the sake of notation: the mesh is made up of cells.

my problem is this :
the trimming area has to be split up into individual smaller
cells bounded by the trimming curve segments. if a cell
is wholly inside the area...then it is output as a whole ,
else it is trivially rejected.

does any body know how this can be done, or is there any algo.
somewhere for doing this.

any help would be appreciated.

thanks,
ani.
--
to get irritated is human, to stay cool, divine.
```

The result POS tagging is shown in Output E

Output E

```
-----pos_tagging:
[('From', 'IN'), (':', ':'), ('ani', 'NN'), ('@', 'NN'), ('ms.uky.edu', 'NN'), ('(', '(':'), ('Aniruddha', 'NNP'), ('B.', 'NNP'), ('Deglurkar', 'NNP'), (')', ':'), ('Subject', 'NN'), (':', ':'), ('help', 'NN'), ('NN'
```

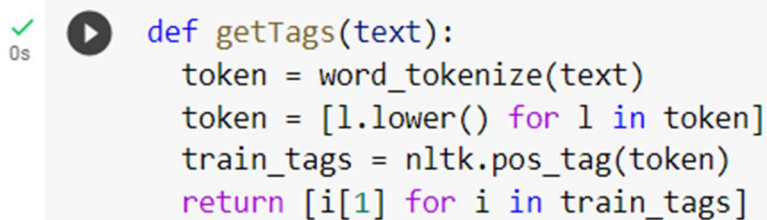
For NLP processing, CountVectorizer and TfidfTransformer will be used to create feature vectors for our machine learning classifier. These will be the indicators for our model in deciding which categories the letter are assigned to.

An alternative to pre-processing of the text is to create tags of the text data. By applying the Part-of-Speech tags for each word in the text, the function can return a list containing all the POS tags. The function firstly tokenizes the text into individual words and converts all the tokens to lowercase using python's built in lower() function. Nltk.pos_tag is a function from the nltk library that tags each token with its corresponding POS tag. The function is pre trained to assign these tags based off relationships between neighbouring words in sentences and context.

Code Snippet C is used to pre-process the raw text.

CountVectorizer is a tool from the sklearn library in python that is used to transform text into a numerical representation so that it can be used in machine learning models. Machine learning models can only take numerical information as its parameters, so this step is necessary for any natural language processing classification task using machine learning. CountVectorizer firstly breaks down the texts into individual words or tokens and then it builds vocabulary of the unique words in the text corpus. By counting the frequency of each word in the document, a value can be encoded and stored within a matrix. This matrix can then be used as a feature vector for a machine learning model.

Code Snippet C



```
def getTags(text):
    token = word_tokenize(text)
    token = [l.lower() for l in token]
    train_tags = nltk.pos_tag(token)
    return [i[1] for i in train_tags]
```

An example of getTags' results is shown in Output F

Output F

```
['IN', ':', 'NN', 'NN', '(', 'JJ', 'NN', 'NN', ')', 'NN', ':', 'NN', ':', 'VBG', 'DT', 'JJ', 'NN', 'IN', 'DT', 'JJ', 'NN',
```

Within the CountVectorizer function from sklearn, a parameter called binary can be manipulated to change the results of the vectorisation [9]. If the value of binary is set to true, then all non-zero counts are set to 1. It can be very useful for discrete probabilistic models that model binary events, such as a binary classification task. This can lead to the reduction of dimensionality of the feature space which then leads to better generalization.

4.3 Classifier results

With the pre-processed data, the pipeline can be iterated to discover how different feature extraction or NLP techniques effect the performance of the machine learning model. 8 different model variations were used and the results of these models are displayed in the table in Table A

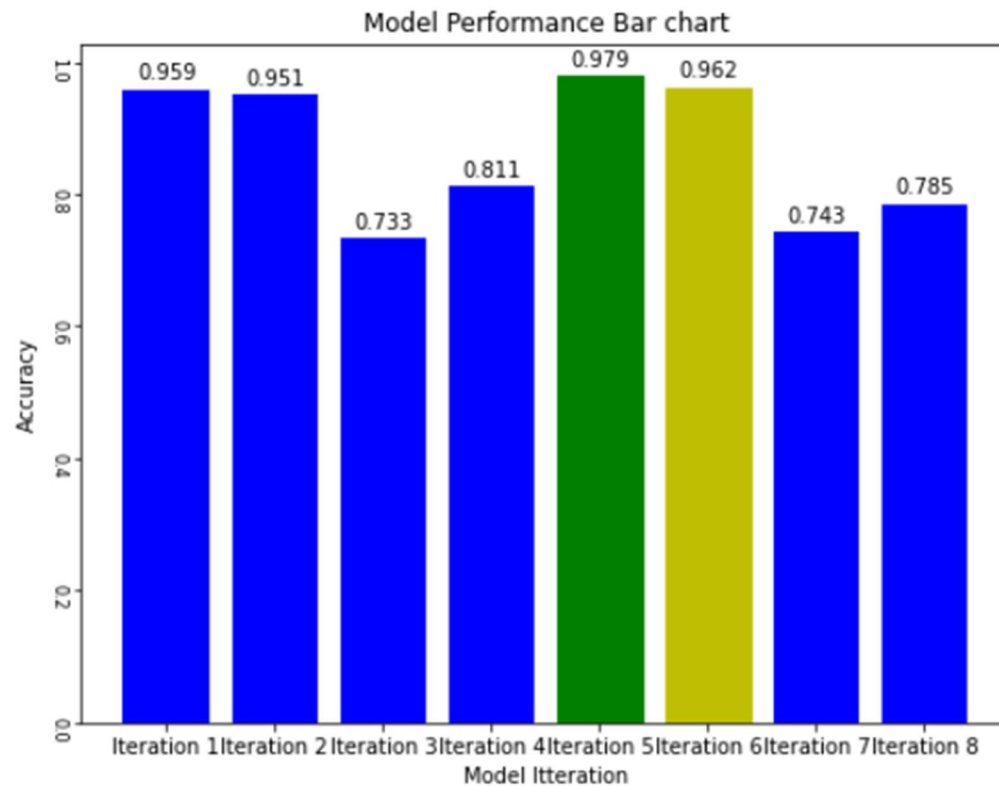
Table A

Iteration Number	Classification model	NLP pre-processing technique	Parameters	Result (accuracy 3dp)	Confusion matrix									
Iteration 1	Logistic regression	processText function using count vectorisation	binary = True	0.959	<table><tr><td></td><td>alt.atheism</td><td>comp.graphics</td></tr><tr><td>alt.atheism</td><td>295</td><td>24</td></tr><tr><td>comp.graphics</td><td>5</td><td>384</td></tr></table>		alt.atheism	comp.graphics	alt.atheism	295	24	comp.graphics	5	384
	alt.atheism	comp.graphics												
alt.atheism	295	24												
comp.graphics	5	384												
Iteration 2	Logistic regression	processText function using count vectorisation	Binary = false	0.951	<table><tr><td></td><td>alt.atheism</td><td>comp.graphics</td></tr><tr><td>alt.atheism</td><td>289</td><td>30</td></tr><tr><td>comp.graphics</td><td>5</td><td>384</td></tr></table>		alt.atheism	comp.graphics	alt.atheism	289	30	comp.graphics	5	384
	alt.atheism	comp.graphics												
alt.atheism	289	30												
comp.graphics	5	384												
Iteration 3	Logistic regression	getTags function using count vectorisation	Binary = True	0.733	<table><tr><td></td><td>alt.atheism</td><td>comp.graphics</td></tr><tr><td>alt.atheism</td><td>223</td><td>96</td></tr><tr><td>comp.graphics</td><td>93</td><td>296</td></tr></table>		alt.atheism	comp.graphics	alt.atheism	223	96	comp.graphics	93	296
	alt.atheism	comp.graphics												
alt.atheism	223	96												
comp.graphics	93	296												
Iteration 4	Logistic regression	getTags function using count vectorisation	Binary = false	0.811	<table><tr><td></td><td>alt.atheism</td><td>comp.graphics</td></tr><tr><td>alt.atheism</td><td>237</td><td>82</td></tr><tr><td>comp.graphics</td><td>52</td><td>337</td></tr></table>		alt.atheism	comp.graphics	alt.atheism	237	82	comp.graphics	52	337
	alt.atheism	comp.graphics												
alt.atheism	237	82												
comp.graphics	52	337												
Iteration 5	SGD Classifier	processText function using count vectorisation	binary = True	0.979	<table><tr><td></td><td>alt.atheism</td><td>comp.graphics</td></tr><tr><td>alt.atheism</td><td>310</td><td>9</td></tr><tr><td>comp.graphics</td><td>6</td><td>383</td></tr></table>		alt.atheism	comp.graphics	alt.atheism	310	9	comp.graphics	6	383
	alt.atheism	comp.graphics												
alt.atheism	310	9												
comp.graphics	6	383												
Iteration 6	SGD Classifier	processText function using count vectorisation	Binary = false	0.962	<table><tr><td></td><td>alt.atheism</td><td>comp.graphics</td></tr><tr><td>alt.atheism</td><td>301</td><td>18</td></tr><tr><td>comp.graphics</td><td>9</td><td>380</td></tr></table>		alt.atheism	comp.graphics	alt.atheism	301	18	comp.graphics	9	380
	alt.atheism	comp.graphics												
alt.atheism	301	18												
comp.graphics	9	380												
Iteration 7	SGD Classifier	getTags function using count vectorisation	Binary = True	0.743	<table><tr><td></td><td>alt.atheism</td><td>comp.graphics</td></tr><tr><td>alt.atheism</td><td>249</td><td>70</td></tr><tr><td>comp.graphics</td><td>118</td><td>271</td></tr></table>		alt.atheism	comp.graphics	alt.atheism	249	70	comp.graphics	118	271
	alt.atheism	comp.graphics												
alt.atheism	249	70												
comp.graphics	118	271												
Iteration 8	SGD Classifier	getTags function using count vectorisation	Binary = false	0.785	<table><tr><td></td><td>alt.atheism</td><td>comp.graphics</td></tr><tr><td>alt.atheism</td><td>192</td><td>127</td></tr><tr><td>comp.graphics</td><td>18</td><td>371</td></tr></table>		alt.atheism	comp.graphics	alt.atheism	192	127	comp.graphics	18	371
	alt.atheism	comp.graphics												
alt.atheism	192	127												
comp.graphics	18	371												

4.4 Evaluation and Error analysis

From the data in Table A, a graph was created to represent the data visually: (created using matplotlib[10])

Figure C



Genially, the logistic regression models performed slightly worse than the SGD classifiers.

The models that used more pre-processing techniques such as miscellaneous character removal and stop word removal had significantly better results than the models that did not include these types of pre-processing techniques.

One possible reason for the SDG models performing better than the LR is because SDG is a lot more computationally efficient and robust to noise when compared to logistic regression. A research paper by Bottou, L investigated the specific characteristics of each model and came to the conclusion that SGD outperformed logistic regression especially when the data had high non-linearity [11]. In our case of the newsletters dataset, we experience high non-linearity as the texts are of random size and have a high variety in the type of writing.

As shown in the graph, iteration 5 performed the best with an overall model accuracy of 0.979. We can look closer into the performance report of this model.

Output G

Accuracy: 0.9759887005649718				
	precision	recall	f1-score	support
alt.atheism	0.98	0.97	0.97	319
comp.graphics	0.97	0.98	0.98	389
accuracy			0.98	708
macro avg	0.98	0.98	0.98	708
weighted avg	0.98	0.98	0.98	708

	alt.atheism	comp.graphics
alt.atheism	309	10
comp.graphics	7	382

The model performed very well by only incorrectly predicting 17 out of the 708 testing data. The SGD classifier misallocated alt.atheism for comp.graphics 10 times and misallocated comp.graphics at alt.atheism 7 times. We can look further into these texts by displaying which articles the model could not predict.

Looking at index 95 the true value is on alt.atheism, however the model predicted it to be comp.graphics.

```
95    From: johnchad@triton.unm.edu (jchadwic)\nSubj...          1    0

From: johnchad@triton.unm.edu (jchadwic)
Subject: Another request for Darwin Fish
Organization: University of New Mexico, Albuquerque
Lines: 11
NNTP-Posting-Host: triton.unm.edu

Hello Gang,

There have been some notes recently asking where to obtain the DARWIN fish.
This is the same question I have and I have not seen an answer on the
net. If anyone has a contact please post on the net or email me.

Thanks,

john chadwick
johnchad@triton.unm.edu
or
```

Perhaps from this text alone, the word “net” was used very frequently which might indicate more towards comp.graphics. Although the Darwin fish does associate with atheism. The Darwin fish statement should have more weights in the decision in This could be due to the fact that the model wasn’t trained on enough information that can deal with “Darwin fish” as this is quite a niche thing in atheism and not many articles would have had it included during the training processes. To solve this problem and allow the machine learning model to accurately predict these types of articles, an increase in the volume of learning potential for the ML classifier’s. In addition, sentiment analysis could be deployed to create greater meaning for each word in the document, hence “Darwin fish” would have significantly more meaning towards the machine learning model, and therefore allowing the model to accurately predict the article’s category.

Accuracy: 0.9505649717514124

	precision	recall	f1-score	support
alt.atheism	0.98	0.91	0.94	319
comp.graphics	0.93	0.99	0.96	389
accuracy			0.95	708
macro avg	0.96	0.95	0.95	708
weighted avg	0.95	0.95	0.95	708

	alt.atheism	comp.graphics
alt.atheism	289	30
comp.graphics	5	384

What was interesting to observe was the logistical regression model incorrectly allocated more atheism to comp.graphics (30) than comp.graphics to alt.atheism (5). This is primarily the trend for all of the models, however, it is because of the class imbalance as mentioned before. The model has been trained on more instances of comp.graphics and therefore can more accurately distinguish them from atheism. An imbalanced dataset may result to a biased machine learning model. There are many ways to handle class imbalance. Under-sampling the majority class, Over-sampling the minority class or by applying SMOTE (synthetic minority oversampling technique)[12]. SMOTE is an algorithm that can generate synthetic samples for minority classes. It can do this by selecting examples that are close in its feature space and drawing a line between the example and feature space.

Cross-validation testing can help prevent bias from getting a “lucky” random iteration when splitting the data into testing and training. By applying a cross validation technique such as k-fold, it is now possible for multiple iterations of the data to be trained and tested to find the mean accuracy and other performance metrics of the models. [13]

The performance and accuracy of the machine learning models can be investigated through their hyperparameters. Specifically, the LR and SGD classifiers have many parameters, such as the alpha value, verbose and many more [14]. A deep analysis into which hyperparameters may best fit this type of classification model can be used to fully optimise the model to deal with this type of data.

5.0 Conclusion and future work

In this research, it was found that the SGD classifier outperformed the logistic regression model in classifying the newsletter articles. With a combination of pre-processing techniques such as stop word removal, unimportant character removal, and feature extraction using TF-IDF, as well as tokenization and count vectorization, the SGD classifier achieved an accuracy of 0.979. This suggests that machine learning and NLP techniques can be powerful tools for text classification tasks, with potential applications in industries such as email sorting or spam filtering. Further work could involve exploring more advanced NLP techniques like sentiment analysis and entity recognition, as well as trying different machine learning models to improve classification performance.

6.0 References and Acknowledgements

- [1] cs.stanford.edu. (n.d.). NLP - overview. [online] Available at: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/nlp/overview_history.html#:~:text=The%20field%20of%20natural%20language.
- [2] OpenAI (2022). ChatGPT: Optimizing Language Models for Dialogue. [online] OpenAI. Available at: <https://openai.com/blog/chatgpt>.
- [3] Tinawi, R. (2020). Text Classification with Python (and some AI Explainability!). [online] Reslan Tinawi. Available at: <https://reslan-tinawi.github.io/2020/05/26/text-classification-using-sklearn-and-nltk.html> [Accessed 13 Mar. 2023].
- [4] Business Analytics. (n.d.). What is noisy text? | Definition from TechTarget. [online] Available at: <https://www.techtarget.com/searchbusinessanalytics/definition/noisy-text> [Accessed 12 Mar. 2023].
- [5] <https://www.capitalone.com/tech/ma/chine-learning/understanding-tf-idf/> [Accessed 11 Mar. 2023].
- [6] Atul Harsha, [online] Available at: <https://www.naukri.com/learning/articles/pos-tagging-in-nlp/> [Accessed 11 Mar. 2023].
- [7] Shalev-Shwartz, S. and Ben-David, S. (2014). Understanding Machine Learning: From Theory to Algorithms. [online] Available at: <https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf>.
- [8] scikit-learn.org. (n.d.). sklearn.pipeline.Pipeline — scikit-learn 0.24.1 documentation. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>.
- [9] Scikit-learn.org. (2018). sklearn.feature_extraction.text.CountVectorizer — scikit-learn 0.20.3 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.
- [10] Matplotlib (2012). Matplotlib: Python plotting — Matplotlib 3.1.1 documentation. [online] Matplotlib.org. Available at: <https://matplotlib.org/>.
- [11] Bottou, L. (2010). Large-Scale Machine Learning with Stochastic Gradient Descent. Proceedings of COMPSTAT'2010, pp.177–186. doi:https://doi.org/10.1007/978-3-7908-2604-3_16.
- [12] Brownlee, J. (2020). SMOTE for Imbalanced Classification with Python. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/>.
- [13] Brownlee, J. (2018). A Gentle Introduction to k-fold Cross-Validation. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/k-fold-cross-validation/>.
- [14] scikit-learn.org. (n.d.). sklearn.linear_model.SGDClassifier — scikit-learn 0.22.2 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html.