HUMAN VALUES TEXT CLASSIFICATION ON SRI SATHYA SAI BABA DISCOURSES USING MACHINE LEARNING

NIHAR VENKATA TATAVARTI

Under the supervision of

SUVAJIT MUKHOPADHYAY

Thesis Report

A thesis submitted in fulfilment of the requirements of
Liverpool John Moores University for the degree of Masters in Data Science

FEBRUARY 2020

ACKNOWLEDGEMENTS

A Humble Offering at the Lotus Feet of Bhagawan Sri Sathya Sai Baba.

I would like to thank my family, friends, supervisor for the support & guidance.

Abstract

There have been many explorations on texts and documents, which required a deeper machine learning methods to classify texts and text sentiment in various applications. Today, businesses focus text categorization on social media, email spam, chats, web pages, customer survey responses with excellent results in natural language processing. There have not been many natural language processing studies on classifying text based on a specific label definition on documents originating from literature pertaining to one specific Individual. For the purpose of this research, the discourses of the orator Bhagawan Sri Sathya Sai Baba that will be referred to as documents are classified across two of the human values referred to as Labels, Love and Right Conduct. The fundamental definition of the label *love* is as per the semantic definition given by orator ‘*Duty without love is deplorable. Duty with love is desirable. Love without duty is Divine’* and the fundamental definition of the label *right conduct* is as per the semantic definition given by orator ‘*Doing work with love is Righteousness’*. Keeping such label definition of the orator’s literature as the foundation, the research attempts to explore the capacity of machine learning and natural language processing. From the look of these definitions and many such similar semantics in documents, various machine learning and natural language processing techniques are used to ascertain the steps involved to rightly classify text across labels. The documents consist of paragraphs segregated across labels based on aforementioned semantic definition which are preprocessed to remove stop words and finding stem of word and lemma of word. Document data sets are then prepared based on natural language processing techniques using Bag of Words, TF-IDF, Word2Vec, Doc2Vec, DeepIR. Various machine learning models are looked at towards finding the higher accuracy. Dimensionality reduction techniques PCA, UMAP and t-SNE are applied and models are further evaluated. All of these models are evaluated, and in the end, a comparison between the various classification models is performed along with formulation of a workflow, which provides better results.

The ultimate aim of this work is to find the workflow that provides best accuracy using the right combination of natural language processing and machine learning as sequential steps to classify the text across human values, love and right conduct, across documents based on discourses of the orator.

Keywords

Machine Learning; Text Categorization; Natural Language Processing;

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

Abbreviation	**Expansion**
ML	Machine Learning
TF-IDF	Term Frequency – Inverse document frequency
PCA	Principal Component Analysis
KNN / k-NN	k-nearest neighbors
UTF	Unicode Transformation Format
MRF	Markov random field
HMM	Hidden Markov Model
SVM	Support Vector Machine
CSV	Comma-separated values
NLTK	Natural Language Toolkit
BOW	Bag Of Words
POS	Parts Of Speech
EDA	Exploratory Data Analysis

NB	Naïve Bayes
XGBoost GBM	Extreme Gradient Boosting
Doc2Vec	Paragraph Vector
DeepIR	Deep Inverse Regression
H20 AutoML	H2O's Auto Machine Learning
UMAP	Uniform Manifold Approximation and Projection
t-SNE	t-distributed stochastic neighbor embedding
PCA	Principal component analysis
SVM rbf	Radial Basis Function (RBF) kernel SVM
CBOW	Continuous Bag of Words
AUC	Area under the ROC Curve
rmse	Root Mean Square Error
mse	Mean squared error
mae	Mean absolute error
rmsle	Root Mean Square Logarithmic Error

LIST OF PYTHON PACKAGES USED

Package	**Description**
numpy	Processing arrays
matplotlib	Plotting graphs
sklearn	Creating ML models, performing cross-validation, generating evaluation metrics
pandas	For creating and working on dataframes
nltk	Natural Language Toolkit
xgboost	Creating ML Extreme Gradient Bossting models, performing cross-validation, generating evaluation metrics
re / regex	For regular expressions
strings	To manage strings
wordcloud	To generate word clouds on corpus
itertools	Functions creating iterators for efficient looping
nltk	Natural language toolkit
genism	topic modeling and natural language processing
keras	preprocessing for text, sequence
h20	H2O's Auto Machine Learning

CHAPTER 1

INTRODUCTION

1.1 Background of the study

Assigning specific text labels according to sentiment or content of the text, this process is known as Text Classification. Businesses are focused on text classification of unstructured data {social media, support tickets, emails, chats, web pages, survey responses, and other areas} for structuring text to enhance decision-making. There has been no interest in Text Classification based on one individual's literature according to the text labels definitions defined by that individual. Human Emotions stem from Human psyche guided by Human Values, which act as the core of Human Existence. This research is focused at finding the workflow and right classifier to classify discourses of Sri Sathya Sai Baba across two human values namely, love and right conduct, being true to the definitions of the human values as defined by Baba during His lifetime. The five Human values are Love, Right Conduct, Truth, Nonviolence and Peace. For this research, the third label will be 'Other' to include Truth, Nonviolence and Peace.

1.2 Problem Statement

To explore and identify the workflow combining natural language processing techniques and machine learning techniques, which can identify the human value the text, is referring to with accuracy more than 70%.

1.3 Definition of Human Value

The Theory of Basic Human Values, developed by Shalom H. Schwartz recognizes ten universal values, which can be organized in four higher order, namely, Openness to change, Self-enhancement, Conservation, Self-transcendence.

This research will solely base on definitions of Human Value as defined by Sri Sathya Sai Baba. To give a general idea, Baba defined five human values. **Love** as per Baba is 'Love is the primordial urge and the basis of creation. Love is God. This love assumes many different forms in the phenomenal world and gives a variety of experiences to individuals. While the forms of love keep changing based on one's relationship, the Principle of Love remains unchanged'. **Right Conduct** is 'Doing work with love is Righteousness'. Peace, Nonviolence and Truth are the remaining three, which are classified as 'Other' for the purpose of research. **Peace** is 'Calmness, endurance, purity, Self-discipline, self-respect'. **Truth** is 'Truth relates to unchanging reality' and **Nonviolence** is 'One should lead a life of moderation and balance. Anything done beyond limits is violence.'

1.4 Aims and Objectives

This aim of the research is to find the workflow and right classifier to accurately classify discourses of Sri Sathya Sai Baba across two human values namely; love and right conduct, being true to the definitions of the human values as defined by Baba during His lifetime.

The objectives for the research are split into three categories, namely, Data Capture and Analysis, Basic natural language processing, Word Embeddings Dimensionality reduction and Modelling.

Data Capture and Analysis consists of building a corpus of Baba's discourse data against three human values, referred to as labels, love, right conduct and other. Label other is a group of corpus that falls under the definition of Truth, Nonviolence and Peace. Converting the corpus into usable format for processing in Python. The phase extends to data cleaning, exploratory data analysis and pre-processing such as removal of stop words, applying stemming and lemmatization techniques.

Basic natural language phase processing converts the pre-processed data into bag of words vectorizer, word level tf-idf, ngram level tf-idf, character level tf-idf, basic pre-trained word-embedding vector using wiki-news-300d-1M.vec. The corpus is converted into test and train datasets or vectors for modelling.

Word embedding phase dives into exploring Gensim Word2Vec, Doc2Vec with 100 dimensions, DeepIR with Word2Vec, Doc2Vec with 300 dimensions and converting the corpus into numpy arrays referred to as datasets or vectors for modelling.

Dimensionality reduction phase uses such as Principal Component analysis and Umap are used on vectors produced in Word embedding phase for modelling.

Modelling phase is where machine learning algorithms Naives Bayes, Logistic regression, SVM linear and rbf, Random Forest, Extreme Gradient Boosting, k-nearest neighbors, Rocchio classification, Gradient Boosting Classifier, Meta Bagging Decision Trees, Ada Boost Classifier, Voting Classifier are applied on vectors from basic natural language phase, word embedding phase and dimensionality reduction phase. The results are analysed and discussed to find the right work flow and classifier that gives the optimum results.

1.5 Scope of the Study

The research is limited to the following constraints.

- The corpus is classified under three labels love, right conduct and other. Peace, Non-voilence and Truth are grouped under label Other.
- In the machine learning approach, only the algorithms listed above in modelling phase are used
- Owing to a limitation in computing resources, a combination of quotes, paragraphs were taken from Baba's discourses.
- In dimensionality reduction, only the techniques listed above in dimensionality reduction phase were used.

1.6 Significance of the Study

The importance of this research work is high as there has been no interest in Text Classification based on one individual's literature according to the text labels definitions defined by that individual. The techniques and work flow can be extended to a corpus of a specific individual for custom text classification where a generic text classification does not yield right results.

1.7 Structure of the Study

This report is divided into chapters in an orderly way as follows:

In Chapter 2, a comprehensive review of the available literatures are discussed. Topics such as text classification, encoding, stemming and lemmatization and machine learning algorithms are discussd.

In Chapter 3, the methodology followed in this research work is described. A detailed discussion of the data capture and analysis , basic natural language phase, word embedding phase, dimensionality reduction phase, and the classification algorithms used are presented. The implementation of the modelling phase is explained. Along with traditional libraries H2O's AutoML is used.

In Chapter 4, the best suited process flow and classifier are presented with results.The study is concluded by presenting the results from the research, suggestions to improve the models and future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Although there is no work to classify text based on one individual's literature according to the text labels definitions defined by that individual, several pieces of research is carried out on text classification, dimensionality reduction techniques on principal component analysis, Uniform Manifold Approximation and Projection, encoding and using machine learning algorithms in text classification

2.2 Encoding.

Different encoding levels are studied in [2], including UTF-8 bytes, characters, words, romanized characters and romanized words to conclude that UTF-8 byte level consistently offers competitive results that linear models remain strong for the text classification task

2.3 Natural Language Processing

Stemming or lemmatization method is a key step in English document processing. Based on three clustering algorithms and two evaluation functions, in [8] a comprehensive study about two stemming algorithms and one lemmatization algorithm are mentioned. According to the experimental result, it shows that the performance is not remarkable, compared with Snowball stemmer and Stanford lemmatization, Porter stemmer can make a better performance in entropy and purity. In [7], word form normalization for the document clustering in a highly inflectional language, Finnish using two word form normalization methods, stemming and lemmatization. On the basis of the experimental results obtained with four hierarchical clustering methods, they conclude that lemmatization is better in conjunction with the clustering of Finnish text than stemming. Lemmas were found better both on the liberal and strict relevance scales, which were established by collapsing the four-point relevance assessment scale.

2.4 Text Classification

The success of machine learning algorithms relies on their capacity to understand complex models and non-linear relationships within data. The limitations of each technique and their application in real-world problems are discussed in [1].

In [3], they have examined the importance of the neighborhood relationship (MRF cliques) among words in an email message for the purpose of spam classification. Results demonstrate that unexpected side effects depending on the neighborhood window size may have larger accuracy impact than the neighborhood relationship effects of the Markov Random Field. Tests of the HMMs on three different combinations (title only; abstract only; and combined set of title and abstract) of document components in [4] showed that, in automatic classification of the OHSUMED documents, the combination of title and abstract information worked better than using a single source of either title or abstract.

Investigation of two widely used approaches for text categorization under the framework of similarity-based learning -- the k-NN algorithm and the Rocchio classifier. Analyzing both algorithms and identify some shortcomings of both. Based on the analysis, a new approach called the kNN model-based algorithm, which combines the strengths of k-NN and the Rocchio classifier in paper [5].

This paper [6] describes the multi-label classification of product review documents using Structured Support Vector Machine. A Structured Support Vector Machine learning paradigm for the multi-label classification of texts from various product reviews is proposed.

2.4 Dimensionality Reduction

In bag-of-words representation of natural language processing, for unsupervised author classification: given unlabeled texts classification problem principal component analysis tends to be a much more efficient way to reduce dimensionality as per [10], and a detailed dimensionality reduction for word embeddings [11] presents approaches to considerably reduces the size of word embeddings while maintaining or improving upon their utility and reducing implementation complexity. Along with principal component analysis, Uniform Manifold Approximation and Projection [13] dwells on a dimension reduction technique that can be used for visualisation similarly to t-SNE, but also for general non-linear dimension reduction with ease of implementation.

2.5 AutoML

The research paper [12] on Automated machine learning explores and states that AutoML significantly improves the efficiency of Machine Learning and has achieved considerable successes in recent years.

2.5 Summary

There have been numerous papers on Text classification. The noteworthy mention has to be [1] giving a detailed idea on solving the problem along with [2] which is required to prepare the corpus. Study conducted in [10], [11] and [13] were used to generate a basic idea on dimensionality reduction, however, and PCA, UMAP were used in raw form in this specific research. As the study in [8] and [7] were not conclusive enough, both stemming and lemmatization was used in the research. The text classification studies mentioned earlier were more to understand the structural approach required to solve problem.

In this work, the focus is on building a corpus, converting the corpus into vectors using natural language processing techniques and dimensionality reduction techniques, building classifiers to accurately predict two human values to arrive at optimal process flow to achieve over 70% accuracy.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

The research undertaken is Exploratory research as there is no work to classify text based on one individual's literature according to the text labels definitions defined by that individual. An attempt is made in finding a solution to the problem using traditional natural language processing and machine learning techniques.

The chapter provides an overview of the methodology followed in the research from inception to culmination.

3.2 Motivation

Bhagwan Sri Sathya Sai Baba has been an influential figure in my life through His teachings and the foundation of those teachings are the Five Human Values, namely TRUTH, RIGHT CONDUCT, PEACE, LOVE and NON-VOILENCE. A complete harmony can only be achieved by adhering to all five human values. In this War & Grief stricken world, His stories, discourses, literature can certainly act as a guiding beacon. Many of learned Data Scientists have done indepth research on Text Categorization in Social media monitoring, Brand monitoring, Voice of Customer however there are very few who have tried to look at how Machine Learning/AI can interpret Human Emotions. Human Emotions stem from Human pysche guided by Human Values which act as the core of Human Existence. At the inception the research started with the problem *Can Machine Learning methods interpret a short story, paragraph, a complete discourse accurately across one of the Human Values {TRUTH, RIGHT CONDUCT, PEACE, LOVE and NON-VOILENCE}?* Which shaped into the problem statement mentioned in section 1.2 with boundaries elaborated in section 1.3 and 1.4 of this research paper.

3.3 Research Method

Research methods are natural language processing techniques and machine learning techniques using from data collection to evaluation. This section covers High level approach used and process work flow followed to arrive at findings

3.3.1 High Level Approach

The below figure elaborates the high level approach to address the problem statement

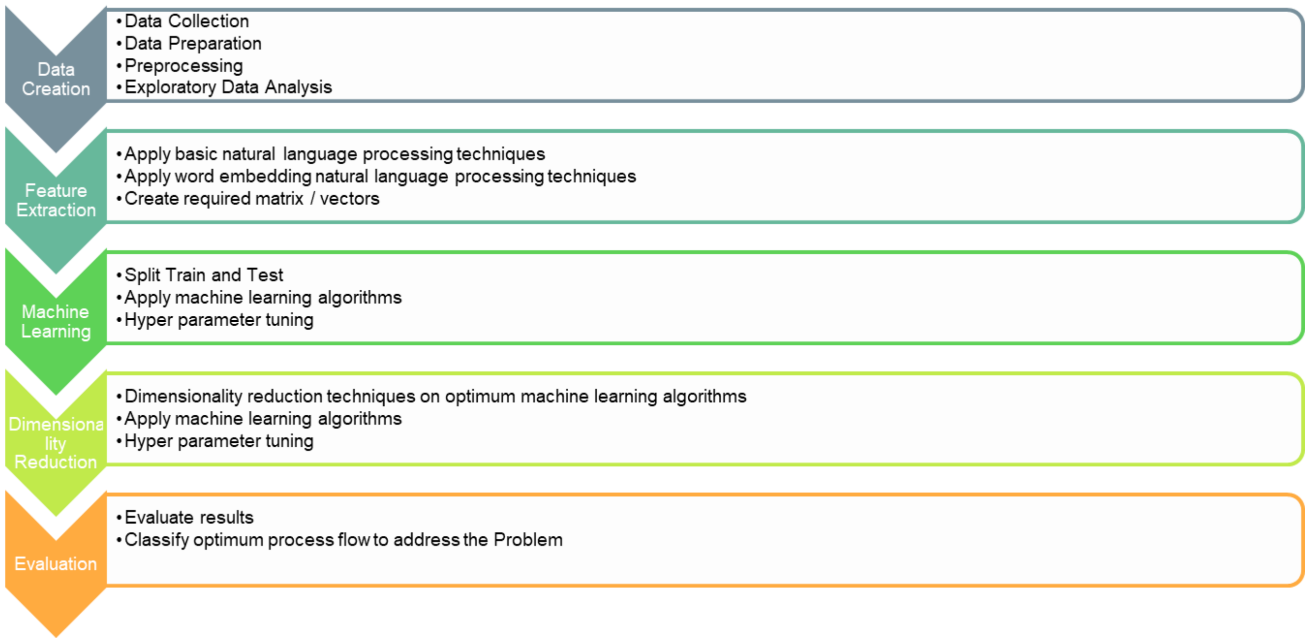

Data Creation
• Data Collection
• Data Preparation
• Preprocessing
• Exploratory Data Analysis
Feature Extraction
• Apply basic natural language processing techniques
• Apply word embedding natural language processing techniques
• Create required matrix / vectors
Machine Learning
• Split Train and Test
• Apply machine learning algorithms
• Hyper parameter tuning
Dimensionality Reduction
• Dimensionality reduction techniques on optimum machine learning algorithms
• Apply machine learning algorithms
• Hyper parameter tuning
Evaluation
• Evaluate results
• Classify optimum process flow to address the Problem

Figure 1 High Level Approach - Research Method

In Data capture & analysis, Data, hence referred to as corpus, is collected from available Sathya Sai Literature and human values are labelled as per the definitions specified in section 1.3 of this paper. The corpus is then prepared for input into python pandas dataframe. The corpus is run through data cleanup procedure using python regular expressions with the *re* module, tokenizing words, removing stop words, converting corpus into lower case, applying stemming and lemmatisation on word corpus and understanding the structure of corpus.

Feature extraction is performed using basic natural language processing and word embedding natural language processing techniques such as Bag of Words, TF-IDF, Word2Vec, Doc2Vec, DeepIR to arrive at required matrix or vectors which will be further used as train and test data as input for machine learning algorithms.

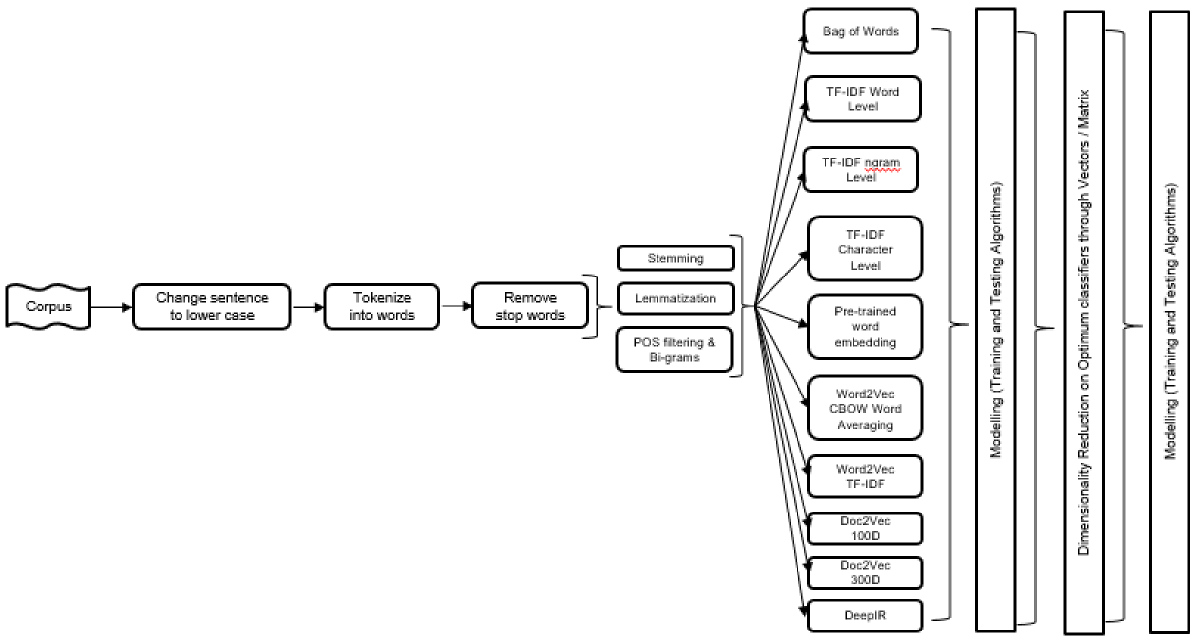
As input into machine learning algorithms, corpus is converted into matrix format or vector format as required, these are split into train and test dataframes. Machine learning algorithms are then trained using Naives Bayes, Logistic regression, SVM linear and rbf, Random Forest, Extreme Gradient Boosting, k-nearest neighbors, Rocchio classification, Gradient Boosting Classifier, Meta Bagging Decision Trees, Ada Boost Classifier, Voting Classifier.

Only the inputs of optimum performing algorithms are selected to apply Dimensionality reduction techniques Principal component analysis, Umap and modelling phase is repeated (*Machine learning algorithms are then trained using Naives Bayes, Logistic regression, SVM linear and rbf, Random Forest, Extreme Gradient Boosting, k-nearest neighbors, Rocchio classification, Gradient Boosting Classifier, Meta Bagging Decision Trees, Ada Boost Classifier, Voting Classifier..)*

Towards the culmination of research, the results are evaluated and based on optimum performing classifiers, a process flow from feature extraction to modelling is called out as a process flow recommendation.

3.4 Methodology (Analysis and Implementation)

This section elaborates on detailed approach used in the research.



Corpus
Change sentence to lower case
Tokenize into words
Remove stop words
Stemming
Lemmatization
POS filtering & Bi-grams
Bag of Words
TF-IDF Word Level
TF-IDF ngram Level
TF-IDF Character Level
Pre-trained word embedding
Word2Vec CBOW Word Averaging
Word2Vec TF-IDF
Doc2Vec 100D
Doc2Vec 300D
DeepIR
Modelling (Training and Testing Algorithms)
Dimensionality Reduction on Optimum classifiers through Vectors / Matrix
Modelling (Training and Testing Algorithms)

Figure 2 Process Flow

The above 'Process Flow' is a pictorial representation of depicts approach used in the research.

3.4.1 Data Capture and Analysis

This section details the steps and procedures executed during data collating and exploratory data analysis.

3.4.1.1 Data Collection

As the research is specific to Sri Sathya Sai Baba discourses and Human values defined by Him, a readymade dataset, that could be used for machine learning was not available. Sai Literature is available for consumption on Sri Sathya Sai Books and Publications Trust [14] and does not need any special approvals to use in as the content of the literature has not been changed or modified or interpreted differently from what it is intended for.

3.1.1 Creating Dataset: Discourses by Human Values

Phrases explaining the human values, study guide on human values based on orator's discources, the vahini series and other literature of the orator was compiled. After compilation, the data is split into two columns [Human Value, Discourse Text]. The Human Value represents Label while paragraph or a quote is discourse text. The data is stored in CSV format with UTF-8 encoding based on learnings from [2]. This step was essential as many researchers on Text classification used ready to use datasets. For this research a dataset had to be prepared which could be imported into python dataframes and lists for further processing.

3.2 Data Cleaning and Preparation

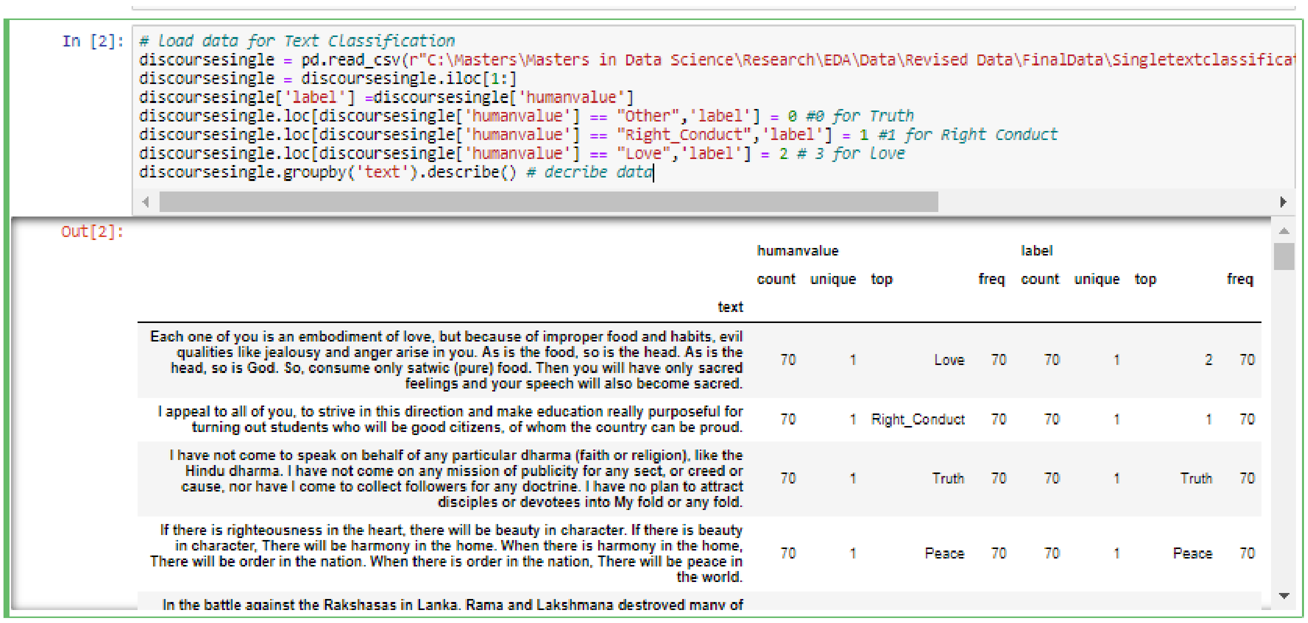
As the data is obtained from books for publication, no grammatical errors or syntactical errors were found. However, apostrophes, hypen, brackets, square brackets, exclamation marks, few Sanskrit words in English had to be cleaned. .

3.2.1 Data Cleaning

In Python, using String and Regular Expressions, corpus was cleaned of punctuation removal, apostrophe removal, removing square brackets, excalamation marks, hypen and brackers. Text changed to lower case. Few regularly occuring sanskrit words Prema, Dharma, Sathya, Shanthi, Ahimsa were converted into Love, Righteousness, Truth, Peace and Nonvoilence. The other sanskrit phrases had English text too, hence these were ignored.

3.2.2 Exploratory Data Analysis

As part of Exploratory Data Analysis we introspect the corpus. The CSV UTF-8 data is imported into a Dataframe with two columns [Human Value, Discourse Text] using Pandas. The corpus had five Classification of Five Human values. However for the research, the corpus was segregated under three labels Love, Right Conduct and Other as discussed in section 1.4.



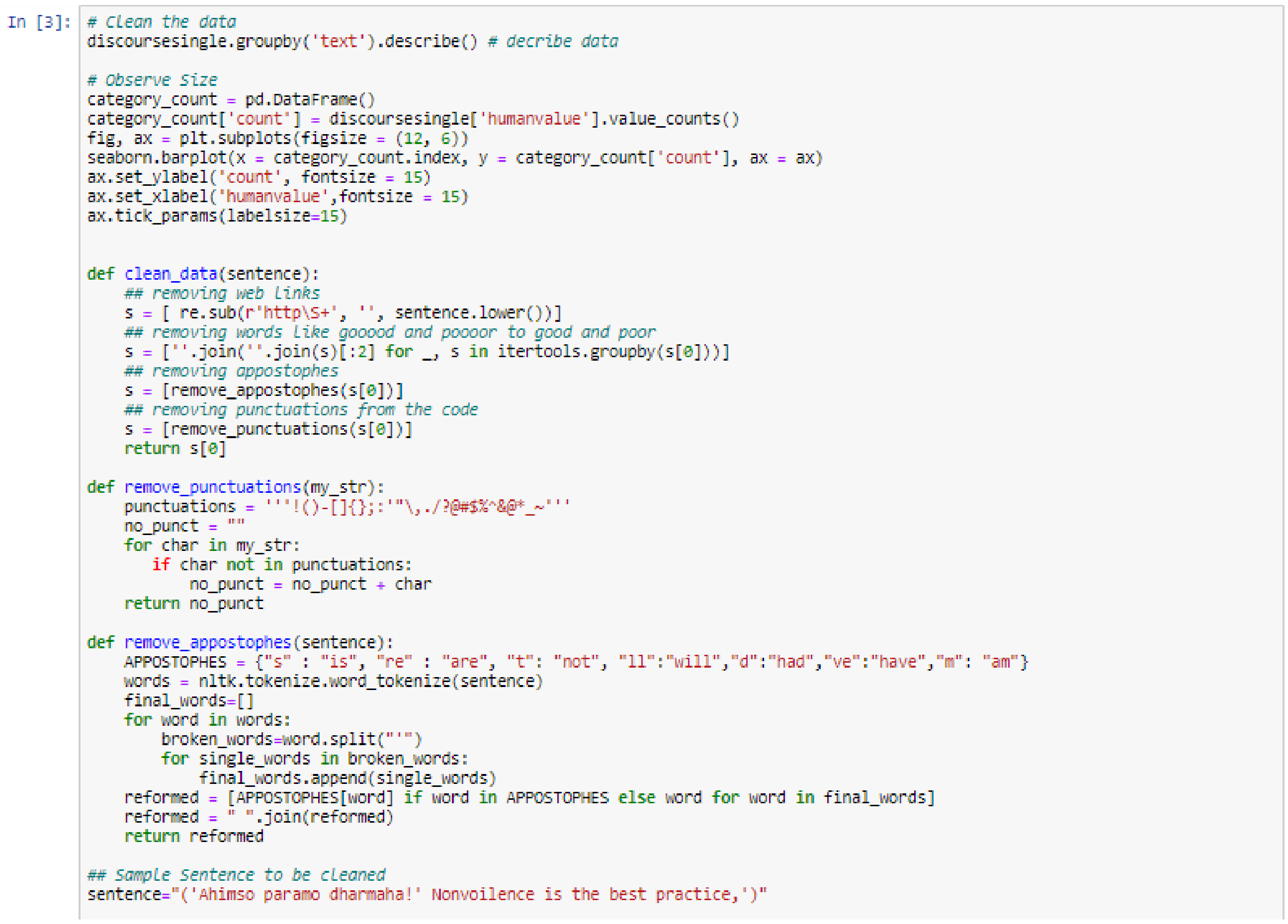
```
In [2]: # Load data for Text Classification
        discoursesingle = pd.read_csv(r"C:\Masters\Masters in Data Science\Research\EDA\Data\Revised Data\FinalData\Singletextclassificat
        discoursesingle = discoursesingle.iloc[1:]
        discoursesingle['label'] =discoursesingle['humanvalue']
        discoursesingle.loc[discoursesingle['humanvalue'] == "Other",'label'] = 0 #0 for Truth
        discoursesingle.loc[discoursesingle['humanvalue'] == "Right_Conduct",'label'] = 1 #1 for Right Conduct
        discoursesingle.loc[discoursesingle['humanvalue'] == "Love",'label'] = 2 # 3 for Love
        discoursesingle.groupby('text').describe() # decribe data
```

Out[2]:

	humanvalue				label			
	count	unique	top	freq	count	unique	top	freq
text								
Each one of you is an embodiment of love, but because of improper food and habits, evil qualities like jealousy and anger arise in you. As is the food, so is the head. As is the head, so is God. So, consume only satwic (pure) food. Then you will have only sacred feelings and your speech will also become sacred.	70	1	Love	70	70	1	2	70
I appeal to all of you, to strive in this direction and make education really purposeful for turning out students who will be good citizens, of whom the country can be proud.	70	1	Right_Conduct	70	70	1	1	70
I have not come to speak on behalf of any particular dharma (faith or religion), like the Hindu dharma. I have not come on any mission of publicity for any sect, or creed or cause, nor have I come to collect followers for any doctrine. I have no plan to attract disciples or devotees into My fold or any fold.	70	1	Truth	70	70	1	Truth	70
If there is righteousness in the heart, there will be beauty in character. If there is beauty in character, There will be harmony in the home. When there is harmony in the home, There will be order in the nation. When there is order in the nation, There will be peace in the world.	70	1	Peace	70	70	1	Peace	70
In the battle against the Rakshasas in Lanka, Rama and Lakshmana destroyed many of								

Figure 3 Describe Corpus

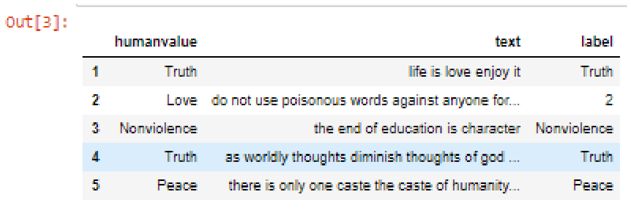
The corpus is cleaned to remove any weblinks, punctuations, apostrophes, remove blank rows (if they exist) and change text to lower case. Functions are written to this effect in Python.



```
In [3]: # Clean the data
        discoursesingle.groupby('text').describe() # decribe data

        # Observe Size
        category_count = pd.DataFrame()
        category_count['count'] = discoursesingle['humanvalue'].value_counts()
        fig, ax = plt.subplots(figsize = (12, 6))
        seaborn.barplot(x = category_count.index, y = category_count['count'], ax = ax)
        ax.set_ylabel('count', fontsize = 15)
        ax.set_xlabel('humanvalue',fontsize = 15)
        ax.tick_params(labelsize=15)


        def clean_data(sentence):
            ## removing web links
            s = [ re.sub(r'http\S+', '', sentence.lower())]
            ## removing words like gooood and poooor to good and poor
            s = [''.join(''.join(s)[:2] for _, s in itertools.groupby(s[0]))]
            ## removing appostophes
            s = [remove_appostophes(s[0])]
            ## removing punctuations from the code
            s = [remove_punctuations(s[0])]
            return s[0]

        def remove_punctuations(my_str):
            punctuations = '''!()-[]{};:'"\,./?@#$%^&@*_~'''
            no_punct = ""
            for char in my_str:
               if char not in punctuations:
                   no_punct = no_punct + char
            return no_punct

        def remove_appostophes(sentence):
            APPOSTOPHES = {"s" : "is", "re" : "are", "t": "not", "ll":"will","d":"had","ve":"have","m": "am"}
            words = nltk.tokenize.word_tokenize(sentence)
            final_words=[]
            for word in words:
                broken_words=word.split("'")
                for single_words in broken_words:
                    final_words.append(single_words)
            reformed = [APPOSTOPHES[word] if word in APPOSTOPHES else word for word in final_words]
            reformed = " ".join(reformed)
            return reformed

        ## Sample Sentence to be cleaned
        sentence="('Ahimso paramo dharmaha!' Nonvoilence is the best practice,')"
```

Figure 4 Clean Data

The corpus is reduced to three labels from five. The corpus for research works on Love, Right Conduct and Other labels



Out[3]:

	humanvalue	text	label
1	Truth	life is love enjoy it	Truth
2	Love	do not use poisonous words against anyone for...	2
3	Nonviolence	the end of education is character	Nonviolence
4	Truth	as worldly thoughts diminish thoughts of god ...	Truth
5	Peace	there is only one caste the caste of humanity...	Peace

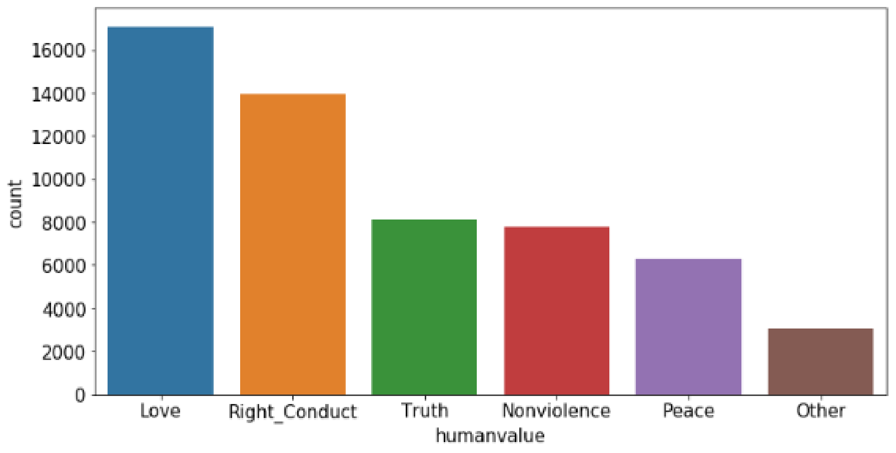
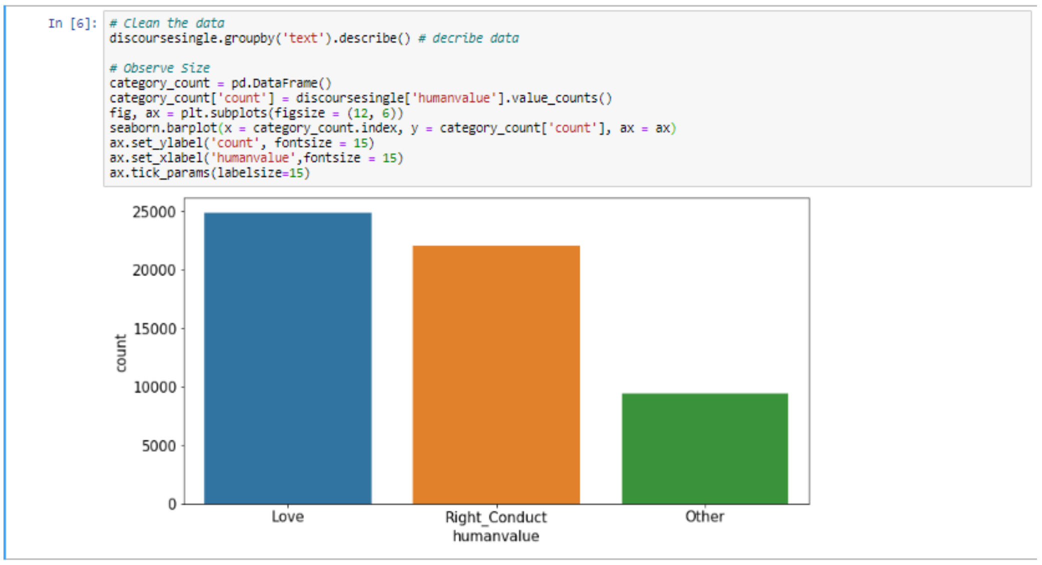


Figure 5 Preliminary Corpus

After reviewing further the label classification as per definition specified in section 1.3 of this paper, corrections were made to the corpus labels and a corpus with combination of paragraphs and quotes was used.



In [6]:
```
# Clean the data
discoursesingle.groupby('text').describe() # decribe data

# Observe Size
category_count = pd.DataFrame()
category_count['count'] = discoursesingle['humanvalue'].value_counts()
fig, ax = plt.subplots(figsize = (12, 6))
seaborn.barplot(x = category_count.index, y = category_count['count'], ax = ax)
ax.set_ylabel('count', fontsize = 15)
ax.set_xlabel('humanvalue',fontsize = 15)
ax.tick_params(labelsize=15)
```

Figure 6 Corpus used in research

3.2.3 Preprocessing

The corpus now has less noise as morphological variation preprocessing steps executed are: corpus converted to lower case, tokenize into words using nltk word tokenize, removing stop words using nltk stopwords, execute stemmer using nltk PorterStemmer, execute lemmatizer using nltk worknetlemmatizer

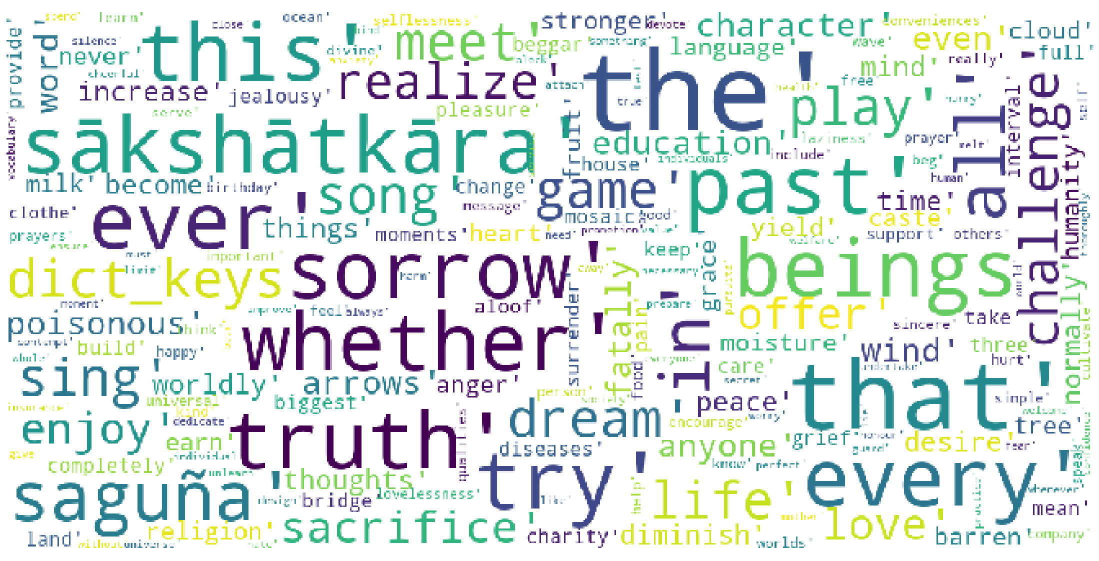


Figure 7 With noise

Noise has reduced and we can see reduced morphological variance in word cloud.

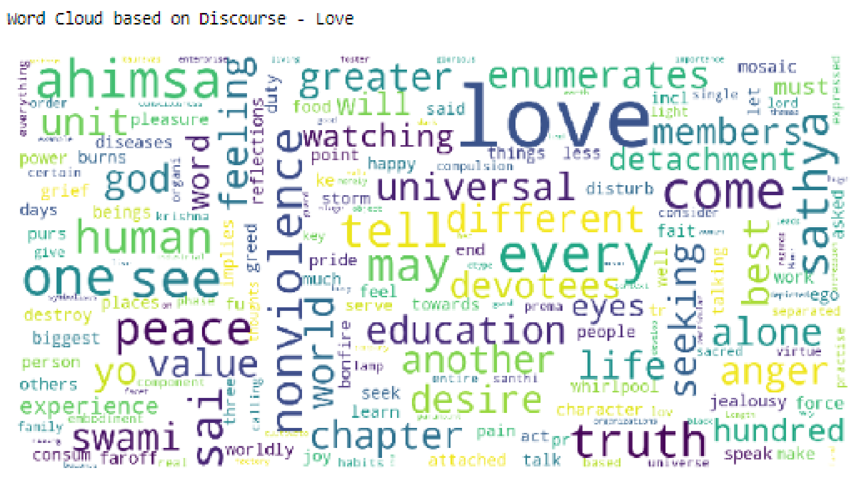


Figure 8 Word Cloud Love

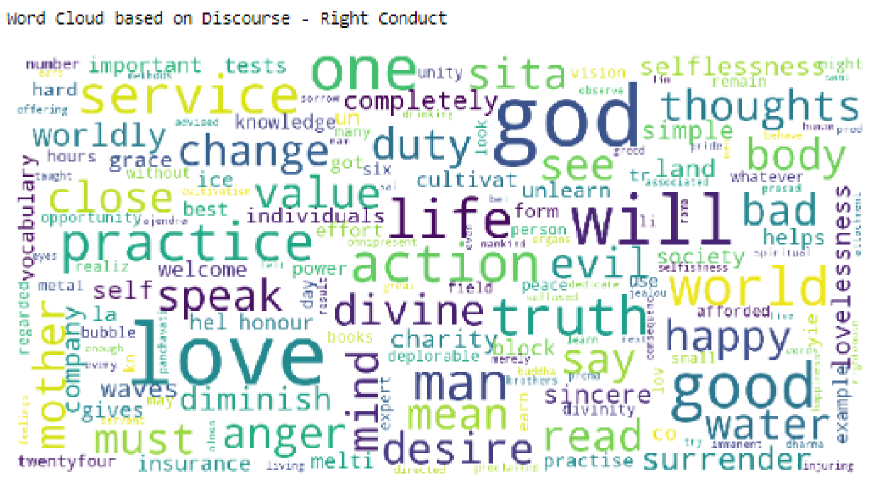


Figure 9 Word Cloud Right Conduct

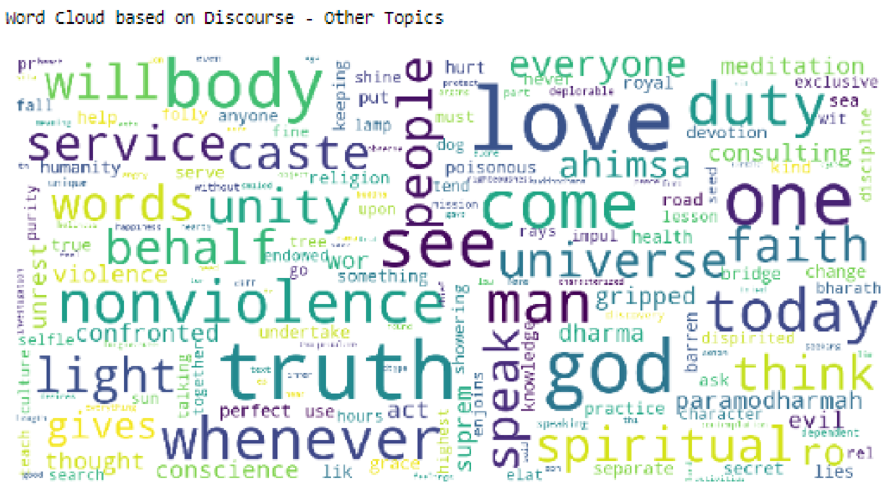
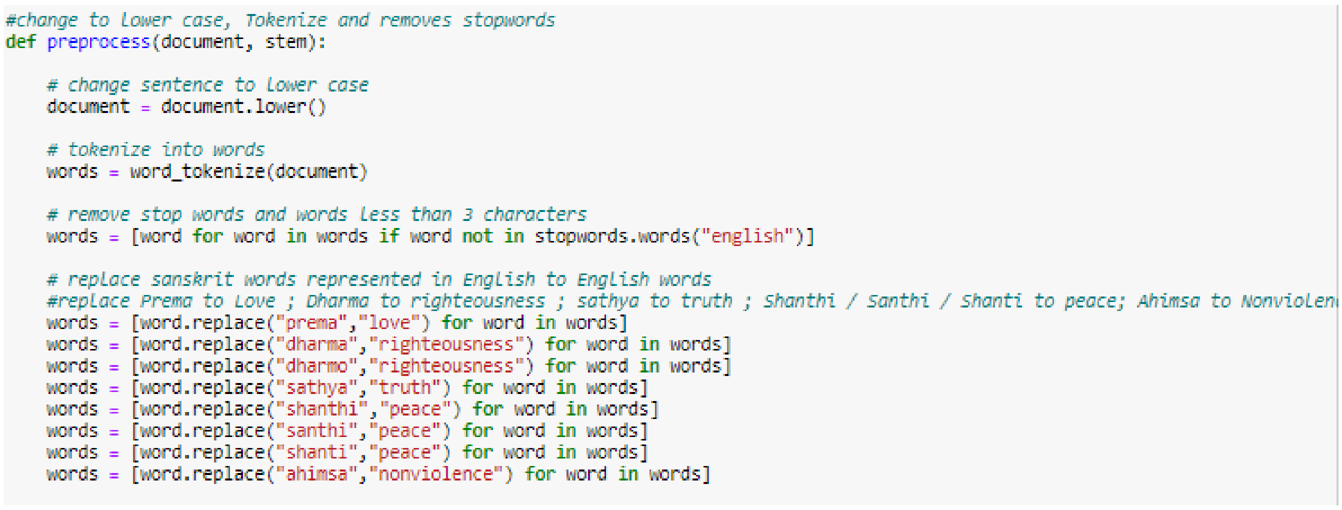


Figure 10 Word Cloud Other

Word Cloud is a data visualization technique which is used for representing corpus based on the frequency or importance of the word. Above mentioned images are word cloud visualization generated based on the labels love, right conduct and other. From the view, it is evident that the word 'love' is is not limited to the corpus labelled Love but is more frequent and stands out a quite often in general manner of speaking.

3.2.4 Exporatory Data Analysis Summary

The Exploratory Data Analysis reveals that the data set has decent size of sentences, words across three labels to be classified. The corpus also contains several Sanskrit words represented English alphabets which are then converted into English language using python replace function. For other Sanskrit words, required English translation is available in corpus and there is no further need for conversion.



```
#change to lower case, Tokenize and removes stopwords
def preprocess(document, stem):

    # change sentence to lower case
    document = document.lower()

    # tokenize into words
    words = word_tokenize(document)

    # remove stop words and words less than 3 characters
    words = [word for word in words if word not in stopwords.words("english")]

    # replace sanskrit words represented in English to English words
    #replace Prema to Love ; Dharma to righteousness ; sathya to truth ; Shanthi / Santhi / Shanti to peace; Ahimsa to Nonviolen
    words = [word.replace("prema","love") for word in words]
    words = [word.replace("dharma","righteousness") for word in words]
    words = [word.replace("dharmo","righteousness") for word in words]
    words = [word.replace("sathya","truth") for word in words]
    words = [word.replace("shanthi","peace") for word in words]
    words = [word.replace("santhi","peace") for word in words]
    words = [word.replace("shanti","peace") for word in words]
    words = [word.replace("ahimsa","nonviolence") for word in words]
```

Figure 11 Converting Sanskrit words to English

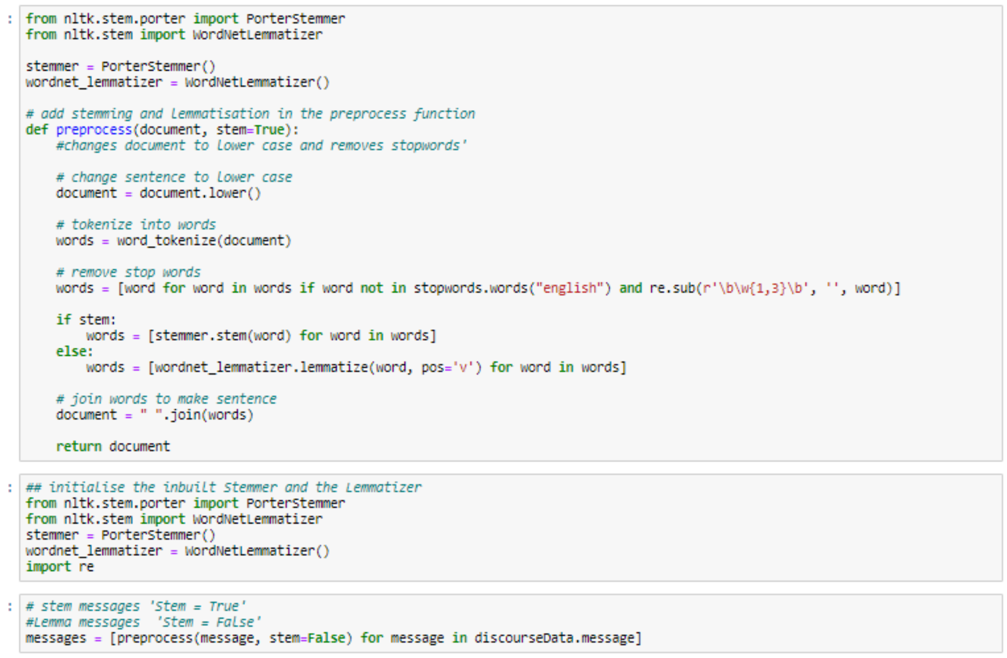
Feature extraction or also known as feature engineering, which is the next step after Data capture and analysis.

3.3 Feature Extraction

This section talks about the basic and advance natural language processing techniques, both lexical processing and syntactic processing that are used in research are discussed.

3.3.1 Overview

The input to this phase are sentences converted into tokenized words {smaller parts} as a fundamental step which are stemmed, lemmatized separately and then joined back as a sentence.



```
: from nltk.stem.porter import PorterStemmer
  from nltk.stem import WordNetLemmatizer

  stemmer = PorterStemmer()
  wordnet_lemmatizer = WordNetLemmatizer()

  # add stemming and lemmatisation in the preprocess function
  def preprocess(document, stem=True):
      #changes document to lower case and removes stopwords'

      # change sentence to lower case
      document = document.lower()

      # tokenize into words
      words = word_tokenize(document)

      # remove stop words
      words = [word for word in words if word not in stopwords.words("english") and re.sub(r'\b\w{1,3}\b', '', word)]

      if stem:
          words = [stemmer.stem(word) for word in words]
      else:
          words = [wordnet_lemmatizer.lemmatize(word, pos='v') for word in words]

      # join words to make sentence
      document = " ".join(words)

      return document
```

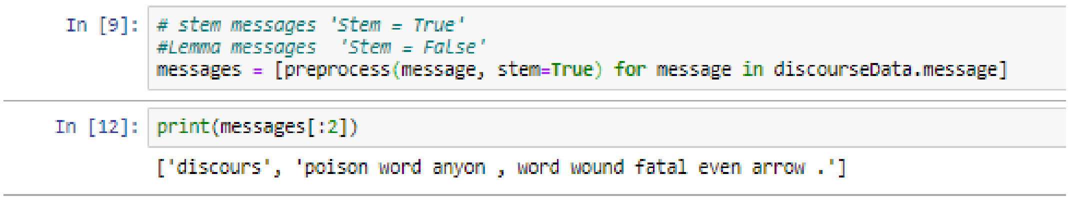
```
: ## initialise the inbuilt Stemmer and the Lemmatizer
  from nltk.stem.porter import PorterStemmer
  from nltk.stem import WordNetLemmatizer
  stemmer = PorterStemmer()
  wordnet_lemmatizer = WordNetLemmatizer()
  import re
```

```
: # stem messages 'Stem = True'
  #Lemma messages  'Stem = False'
  messages = [preprocess(message, stem=False) for message in discourseData.message]
```

Figure 12 Stemmer and Lemmatizer

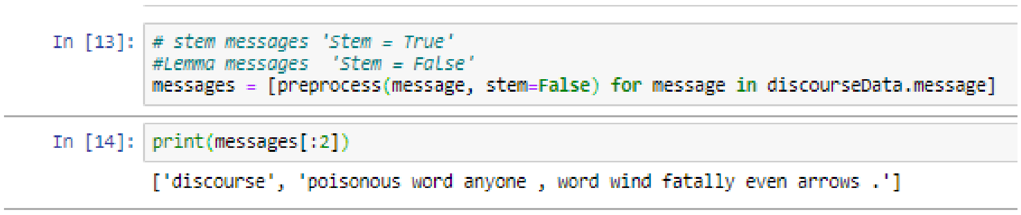
Text normalization is achieved using both stemming and lemmatization. While stemming reduces word to its root form which may or may not resemble an actual word, where as lemmatization converts into an actual word. Porterstemmer is used as it is simple and efficient. WordNet lemmatizer is used as it uses WordNet database to fetch lemmas of words.

A sentence '*Do not use poisonous words against anyone, for words wound more fatally than even arrows*' after Tokenization translates into *['poison', 'word', 'anyone', 'word', 'wound', 'fatal', 'even', 'arrow']*



```
In [9]:  # stem messages 'Stem = True'
         #Lemma messages  'Stem = False'
         messages = [preprocess(message, stem=True) for message in discourseData.message]

In [12]: print(messages[:2])

         ['discours', 'poison word anyon , word wound fatal even arrow .']
```

Figure 13 Stemming



```
In [13]: # stem messages 'Stem = True'
         #Lemma messages  'Stem = False'
         messages = [preprocess(message, stem=False) for message in discourseData.message]

In [14]: print(messages[:2])

         ['discourse', 'poisonous word anyone , word wind fatally even arrows .']
```

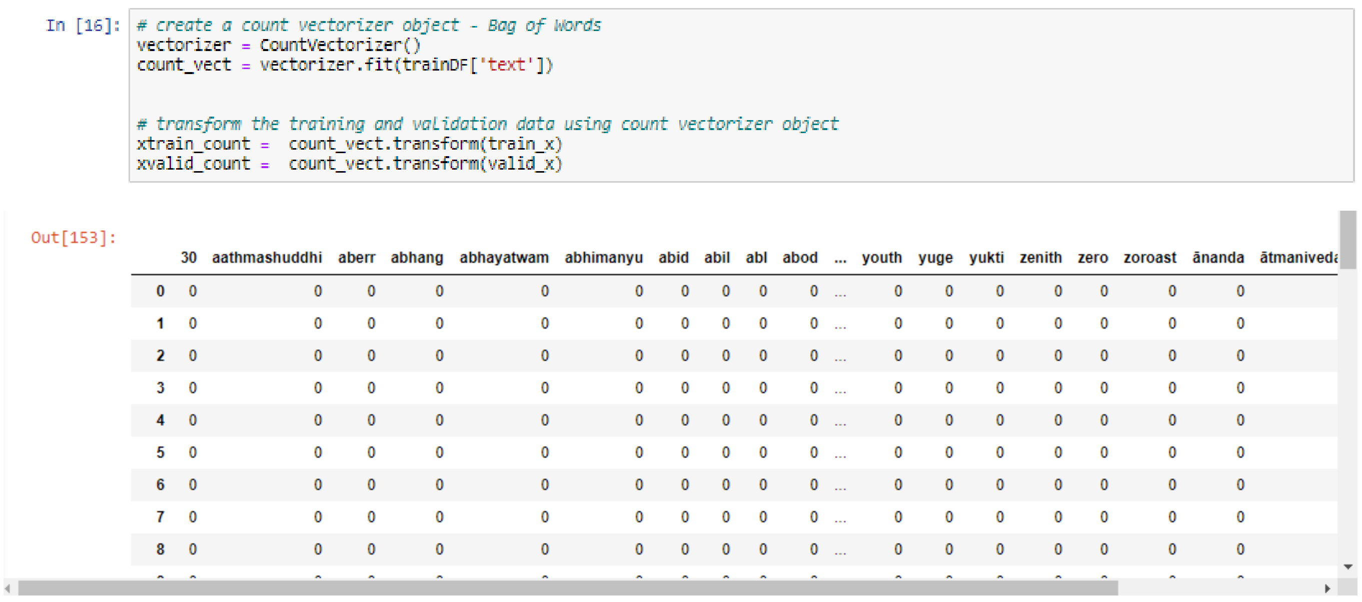
Figure 14 Lemmatization

Applying Porterstemmer converts the sentence into *poison word anyon , word wound fatal ev en arrow* and while WordNet lemmatizer translates into *poisonous word anyone , word wind fatally even arrows*. The difference in this depiction is the word *anyone* which stemmer reduc es to it's root 'anyon' while lemmatizer retains the actual word.

After converting the corpus through tokenization followed by stemming and lemmatization, applied separately, the feature extraction techniques used in research are Bag of Words, TF-IDF, creating matrix using pre-trained word embedding vector using wiki-news-300d-1M, using Word2Vec keyed vector GoogleNews-vectors-negative300.bin.gz and deriving continuous bad of words based on word averaging, using Word2Vec keyed vector GoogleNews-vectors-negative300.bin.gz and deriving continuous bad of words based on IF-IDF, Doc2Vec of 100 dimensions, Doc2Vec of 300 dimensions and finally Deep IR Technique developed by Matt Taddy [13].

3.3.2 Bag of Words

Bag of words (BOW) is a representation model of vector space of the corpus. The representation of all the words that occur in it (after removing the stopwords), where the sequence of occurrence does not matter. Using sklearn.feature_extraction.text.CountVectorizer the corpus is converted into a matrix of token counts. This implementation produces a sparse representation of the counts.



```
In [16]: # create a count vectorizer object - Bag of Words
         vectorizer = CountVectorizer()
         count_vect = vectorizer.fit(trainDF['text'])

         # transform the training and validation data using count vectorizer object
         xtrain_count =  count_vect.transform(train_x)
         xvalid_count =  count_vect.transform(valid_x)
```

Out[153]:

	30	aathmashuddhi	aberr	abhang	abhayatwam	abhimanyu	abid	abil	abl	abod	...	youth	yuge	yukti	zenith	zero	zoroast	ānanda	ātmaniveda
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	

Figure 15 : BOW Vector

3.3.3 TF-IDF

Term Frequency – Inverse document frequency takes into the account the importance of each word.

The formula to calculate TF-IDF weight of a term in a document is:

$$tf_{t,d} = \text{frequency of term 't' in document 'd' / total terms in document 'd'}$$

$$idf_t = \log (\text{total number of documents / total documents that have the term 't'})$$

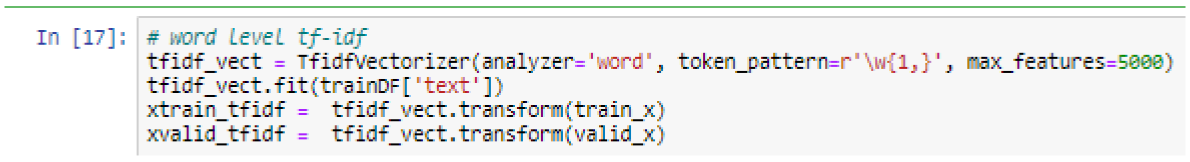
The log in the above formula is with base 10. The tf-idf score for any term in a document is just the product of these two terms:

$$tf\text{–}idf = tf_{t,d} * idf_t$$

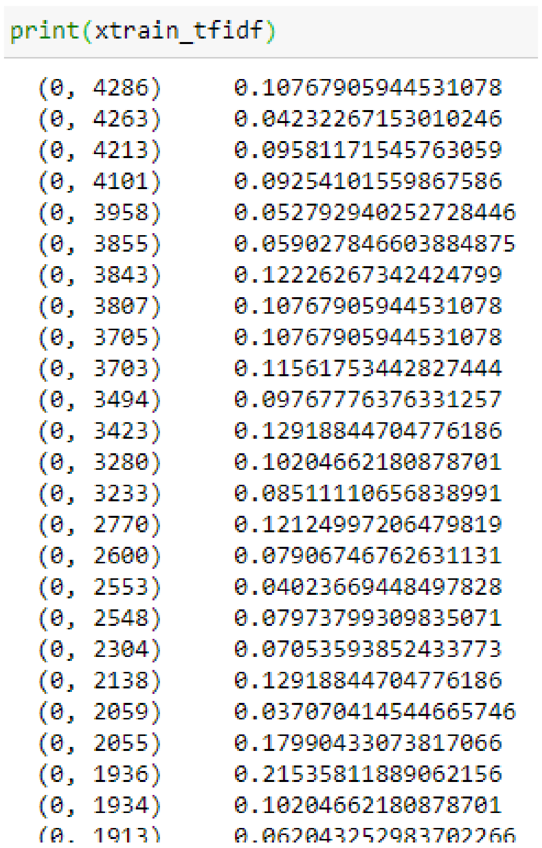
Higher weights are assigned to terms that are present frequently in a document and which are rare among all documents. On the other hand, a low score is assigned to terms, which are common across all documents.

TF-IDF – Word Embedding

Using TfidfVectorizer of sklearn.feature_extraction.text created a TF-IDF on all words. All words are formed by 26 (or 52 if including either upper and lower case character, or even more if including special characters). This leaves any new word introductions or characters such as Sanskrit words in English. Max_features parameter is used to limit the vocabulary to the most feature frequency.



```
In [17]: # word level tf-idf
         tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', max_features=5000)
         tfidf_vect.fit(trainDF['text'])
         xtrain_tfidf =  tfidf_vect.transform(train_x)
         xvalid_tfidf =  tfidf_vect.transform(valid_x)
```

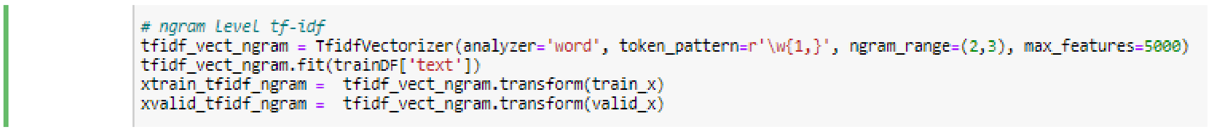


```
print(xtrain_tfidf)

  (0, 4286)	0.10767905944531078
  (0, 4263)	0.04232267153010246
  (0, 4213)	0.09581171545763059
  (0, 4101)	0.09254101559867586
  (0, 3958)	0.052792940252728446
  (0, 3855)	0.059027846603884875
  (0, 3843)	0.12226267342424799
  (0, 3807)	0.10767905944531078
  (0, 3705)	0.10767905944531078
  (0, 3703)	0.11561753442827444
  (0, 3494)	0.09767776376331257
  (0, 3423)	0.12918844704776186
  (0, 3280)	0.10204662180878701
  (0, 3233)	0.08511110656838991
  (0, 2770)	0.12124997206479819
  (0, 2600)	0.07906746762631131
  (0, 2553)	0.04023669448497828
  (0, 2548)	0.07973799309835071
  (0, 2304)	0.07053593852433773
  (0, 2138)	0.12918844704776186
  (0, 2059)	0.037070414544665746
  (0, 2055)	0.17990433073817066
  (0, 1936)	0.21535811889062156
  (0, 1934)	0.10204662180878701
  (0, 1913)	0.062043252983702266
```

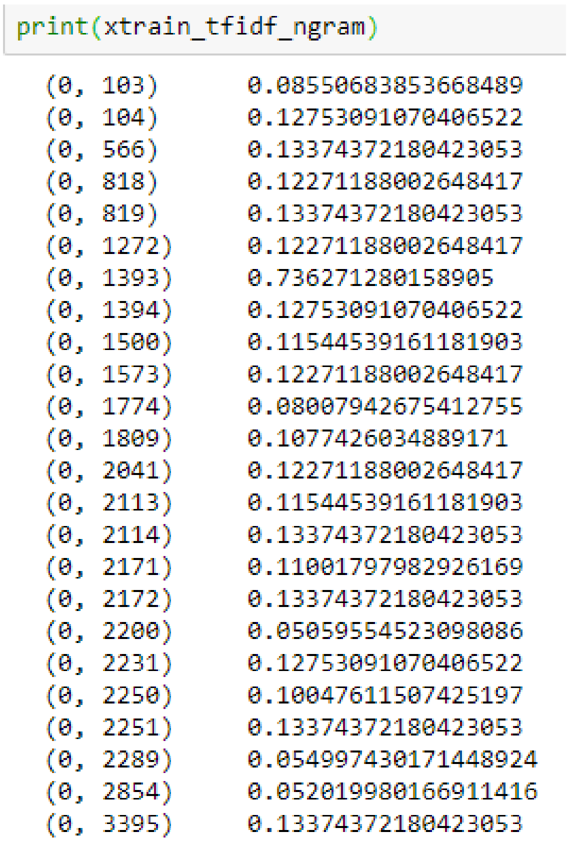
Figure 16 : TF-IDF Word Embedding Vector

TF-IDF – n gram Embedding

Setting the ngram_range=(2,3) in TfidfVectorizer, The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that min_n <= n <= max_n will be used. The minimum range is 2 and maximum range is 3. 1-gram is already covered in TF-IDF – Word Embedding word embedding. Max_features parameter is used to limit the vocabulary to the most feature frequency.



```
# ngram level tf-idf
tfidf_vect_ngram = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', ngram_range=(2,3), max_features=5000)
tfidf_vect_ngram.fit(trainDF['text'])
xtrain_tfidf_ngram =  tfidf_vect_ngram.transform(train_x)
xvalid_tfidf_ngram =  tfidf_vect_ngram.transform(valid_x)
```



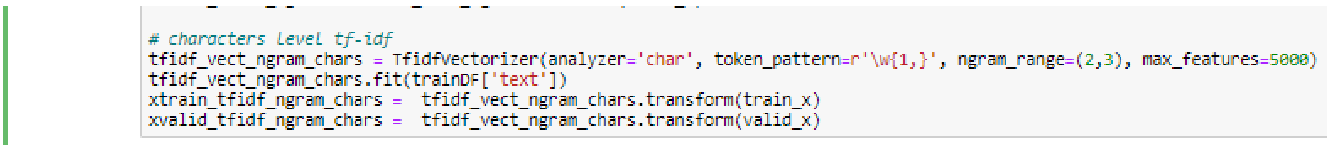
```
print(xtrain_tfidf_ngram)
```

```
  (0, 103)	0.08550683853668489
  (0, 104)	0.12753091070406522
  (0, 566)	0.13374372180423053
  (0, 818)	0.12271188002648417
  (0, 819)	0.13374372180423053
  (0, 1272)	0.12271188002648417
  (0, 1393)	0.736271280158905
  (0, 1394)	0.12753091070406522
  (0, 1500)	0.11544539161181903
  (0, 1573)	0.12271188002648417
  (0, 1774)	0.08007942675412755
  (0, 1809)	0.1077426034889171
  (0, 2041)	0.12271188002648417
  (0, 2113)	0.11544539161181903
  (0, 2114)	0.13374372180423053
  (0, 2171)	0.11001797982926169
  (0, 2172)	0.13374372180423053
  (0, 2200)	0.05059554523098086
  (0, 2231)	0.12753091070406522
  (0, 2250)	0.10047611507425197
  (0, 2251)	0.13374372180423053
  (0, 2289)	0.054997430171448924
  (0, 2854)	0.052019980166911416
  (0, 3395)	0.13374372180423053
```

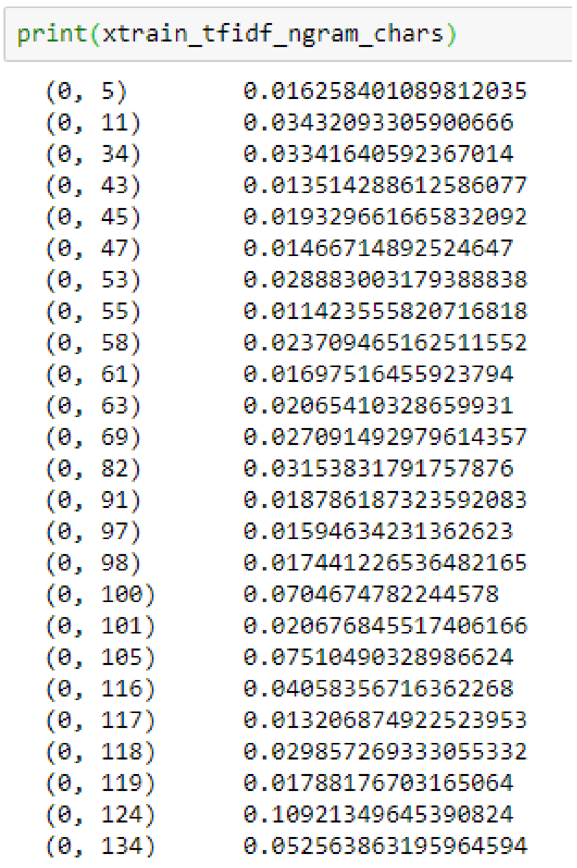
Figure 17 : TF-IDF Word N-Gram Vector

TF-IDF – Character Embedding

Character embedding solves the problem of Sanskrit words, or new English words in dictionary. We set analyzer to 'char' and set ngram_range=(2,3) in TfidfVectorizer



```
# characters level tf-idf
tfidf_vect_ngram_chars = TfidfVectorizer(analyzer='char', token_pattern=r'\w{1,}', ngram_range=(2,3), max_features=5000)
tfidf_vect_ngram_chars.fit(trainDF['text'])
xtrain_tfidf_ngram_chars =  tfidf_vect_ngram_chars.transform(train_x)
xvalid_tfidf_ngram_chars =  tfidf_vect_ngram_chars.transform(valid_x)
```



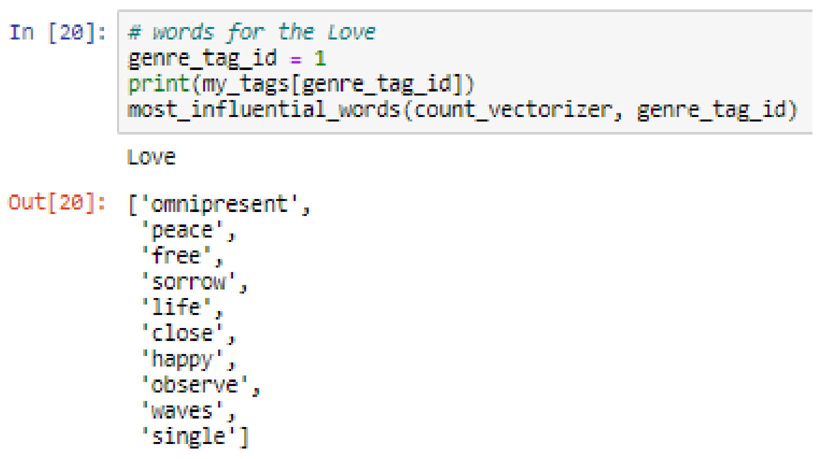
```
print(xtrain_tfidf_ngram_chars)
```

```
  (0, 5)	0.01625840108981203 5
  (0, 11)	0.0343209330590066 6
  (0, 34)	0.0334164059236701 4
  (0, 43)	0.013514288612586077
  (0, 45)	0.019329661665832092
  (0, 47)	0.0146671489252464 7
  (0, 53)	0.028883003179388838
  (0, 55)	0.011423555820716818
  (0, 58)	0.023709465162511552
  (0, 61)	0.0169751645592379 4
  (0, 63)	0.0206541032865993 1
  (0, 69)	0.027091492979614357
  (0, 82)	0.0315383179175787 6
  (0, 91)	0.018786187323592083
  (0, 97)	0.0159463423136262 3
  (0, 98)	0.017441226536482165
  (0, 100)	0.070467478224457 8
  (0, 101)	0.020676845517406166
  (0, 105)	0.0751049032898662 4
  (0, 116)	0.0405835671636226 8
  (0, 117)	0.013206874922523953
  (0, 118)	0.029857269333055332
  (0, 119)	0.0178817670316506 4
  (0, 124)	0.1092134964539082 4
  (0, 134)	0.052563863195964594
```

Figure 18 : TF-IDF Character N-Gram Vector

3.3.4 Word Embeddings

So far, we have considered frequency based embeddings. Now we use word embeddings to consider and create prediction based vectors. Word Embeddings map the words and phrases in vocabulary to to vectors of real numbers. Most influential, most similar words can are considered in the resultant vector. Word embeddings provide better word representation capturing the right semantic and syntactic word relationship

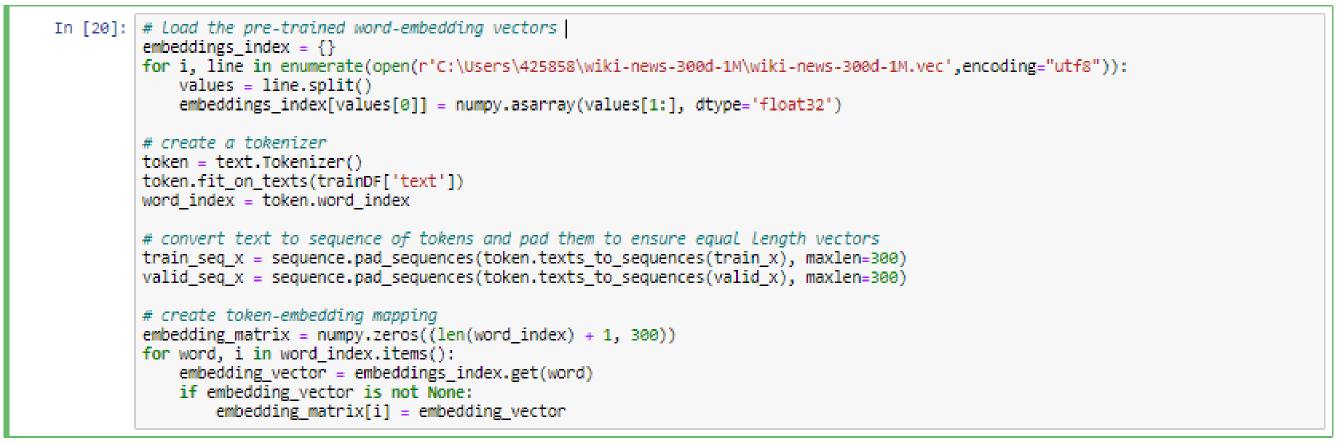


```
In [20]: # words for the Love
         genre_tag_id = 1
         print(my_tags[genre_tag_id])
         most_influential_words(count_vectorizer, genre_tag_id)

         Love

Out[20]: ['omnipresent',
          'peace',
          'free',
          'sorrow',
          'life',
          'close',
          'happy',
          'observe',
          'waves',
          'single']
```

Figure 19 Word Embeddings Most Influential Words

Pre-trained word embedding vector

wiki-news-300d-1M.vec [16] contains 1 million word vectors trained on Wikipedia 2017 MBC webbase corpus and statmt.org news dataset. To create a numpy array on the corpus, fastText word vector wiki-news-300d-1M.vec [16] is used. We create vectors of length 300.

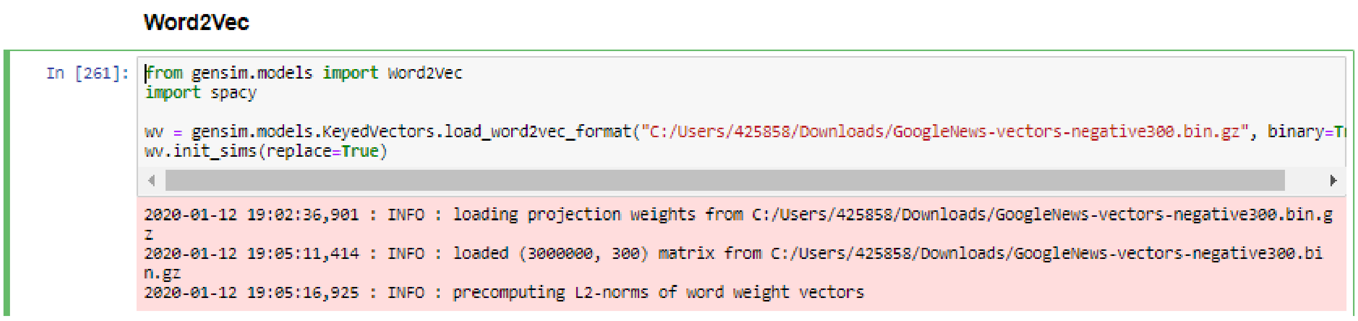


```
In [20]: # Load the pre-trained word-embedding vectors
         embeddings_index = {}
         for i, line in enumerate(open(r'C:\Users\425858\wiki-news-300d-1M\wiki-news-300d-1M.vec',encoding="utf8")):
             values = line.split()
             embeddings_index[values[0]] = numpy.asarray(values[1:], dtype='float32')

         # create a tokenizer
         token = text.Tokenizer()
         token.fit_on_texts(trainDF['text'])
         word_index = token.word_index

         # convert text to sequence of tokens and pad them to ensure equal length vectors
         train_seq_x = sequence.pad_sequences(token.texts_to_sequences(train_x), maxlen=300)
         valid_seq_x = sequence.pad_sequences(token.texts_to_sequences(valid_x), maxlen=300)

         # create token-embedding mapping
         embedding_matrix = numpy.zeros((len(word_index) + 1, 300))
         for word, i in word_index.items():
             embedding_vector = embeddings_index.get(word)
             if embedding_vector is not None:
                 embedding_matrix[i] = embedding_vector
```

Figure 17 Pre-trained word embedding vector – fastText

Word2Vec

One of the most popular algorithms in computing embeddings is Word2Vec [17] [18] [19]. This technique contains a mini neural network that tries to learn a language model. We use genism, which is an library for topic modelling and natural language processing. GoogleNews-vectors-negative300.bin.gz containing pre-trained Google News corpus of 3 million 300-dimension English word vectors is used to train the corpus.


Word2Vec

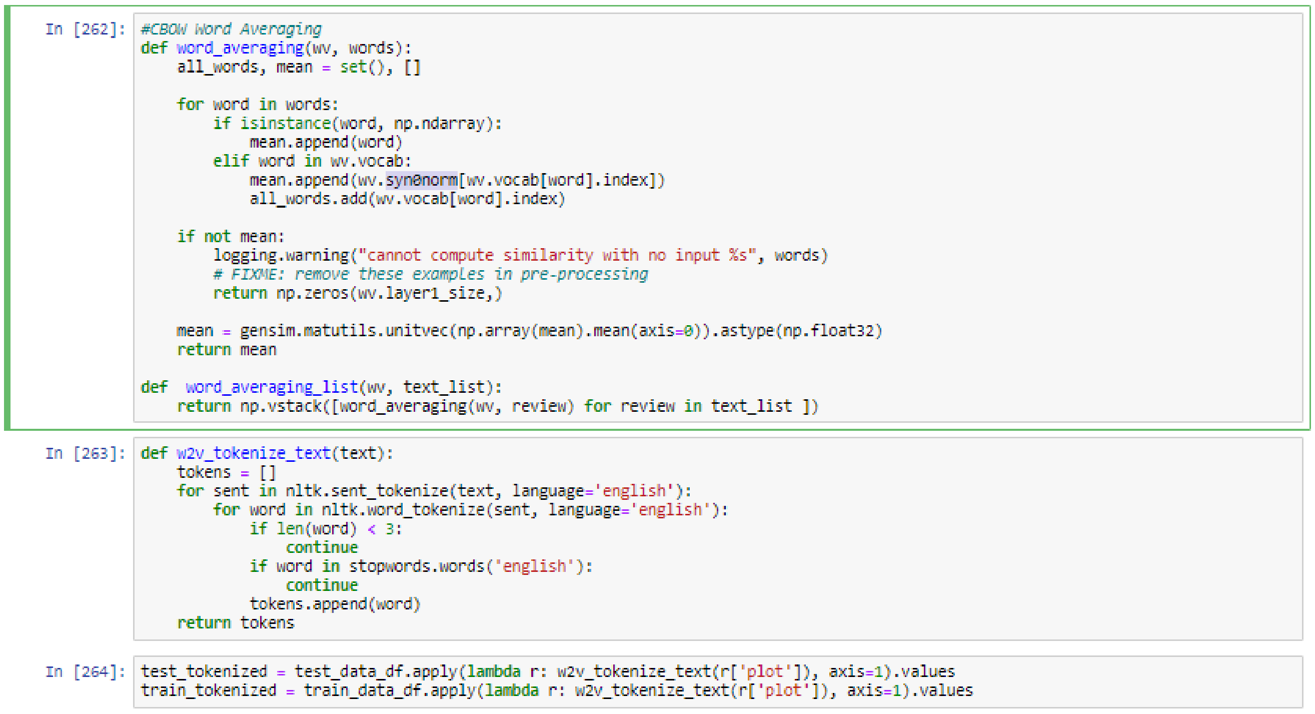
```
In [261]: from gensim.models import Word2Vec
          import spacy

          wv = gensim.models.KeyedVectors.load_word2vec_format("C:/Users/425858/Downloads/GoogleNews-vectors-negative300.bin.gz", binary=T
          wv.init_sims(replace=True)
```

```
2020-01-12 19:02:36,901 : INFO : loading projection weights from C:/Users/425858/Downloads/GoogleNews-vectors-negative300.bin.g
z
2020-01-12 19:05:11,414 : INFO : loaded (3000000, 300) matrix from C:/Users/425858/Downloads/GoogleNews-vectors-negative300.bi
n.gz
2020-01-12 19:05:16,925 : INFO : precomputing L2-norms of word weight vectors
```

Figure 20 Load GoogleNews-vectors-negative300.bin.gz

CBOW Word Averaging

Continuous Bag Of Words, referred to as CBOW, predicts the probability of word in the given context. CBOW takes the context terms as the input and predicts the target or given term. The corpus, tokenized and lemmatized to be parsed through a function which computes average word vector of multiple documents. We refer to this as CBOW Word Averaging vector in the research paper.



```
In [262]: #CBOW Word Averaging
          def word_averaging(wv, words):
              all_words, mean = set(), []

              for word in words:
                  if isinstance(word, np.ndarray):
                      mean.append(word)
                  elif word in wv.vocab:
                      mean.append(wv.syn0norm[wv.vocab[word].index])
                      all_words.add(wv.vocab[word].index)

              if not mean:
                  logging.warning("cannot compute similarity with no input %s", words)
                  # FIXME: remove these examples in pre-processing
                  return np.zeros(wv.layer1_size,)

              mean = gensim.matutils.unitvec(np.array(mean).mean(axis=0)).astype(np.float32)
              return mean

          def  word_averaging_list(wv, text_list):
              return np.vstack([word_averaging(wv, review) for review in text_list ])
```

```
In [263]: def w2v_tokenize_text(text):
              tokens = []
              for sent in nltk.sent_tokenize(text, language='english'):
                  for word in nltk.word_tokenize(sent, language='english'):
                      if len(word) < 3:
                          continue
                      if word in stopwords.words('english'):
                          continue
                      tokens.append(word)
              return tokens
```

```
In [264]: test_tokenized = test_data_df.apply(lambda r: w2v_tokenize_text(r['plot']), axis=1).values
          train_tokenized = train_data_df.apply(lambda r: w2v_tokenize_text(r['plot']), axis=1).values
```

Figure 21 CBOW

TF-IDF embedding vectorizer

Tokenized words from Corpus such as bi-grams, part of speech are filtered. TFIDF model is built to compute to compute each word's idf as its weight. Term Frequency – Inverse document frequency takes into the account the importance of each word.

The formula to calculate TF-IDF weight of a term in a document is:

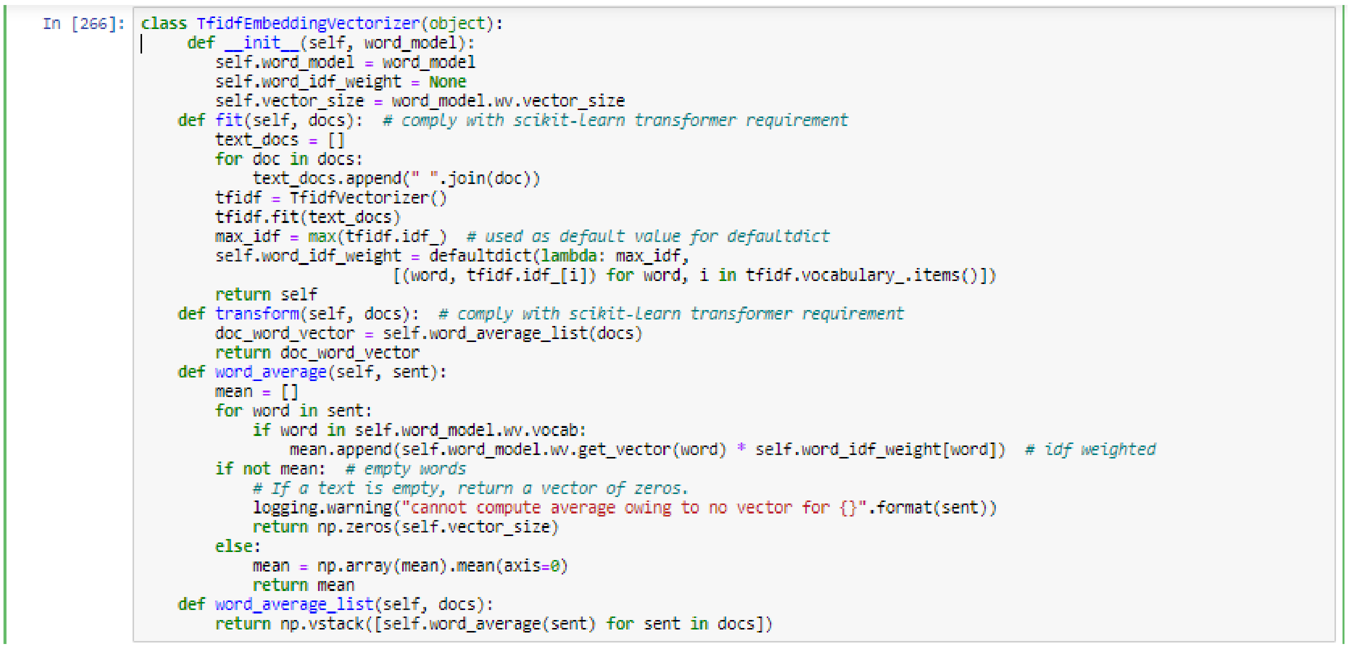
$$tf_{t,d} = \text{frequency of term 't' in document 'd' / total terms in document 'd'}$$

$$idf_t = \log\text{(total number of documents / total documents that have the term 't')}$$

The log in the above formula is with base 10. The tf-idf score for any term in a document is just the product of these two terms:

$$tf\text{–}idf = tf_{t,d} * idf_t$$

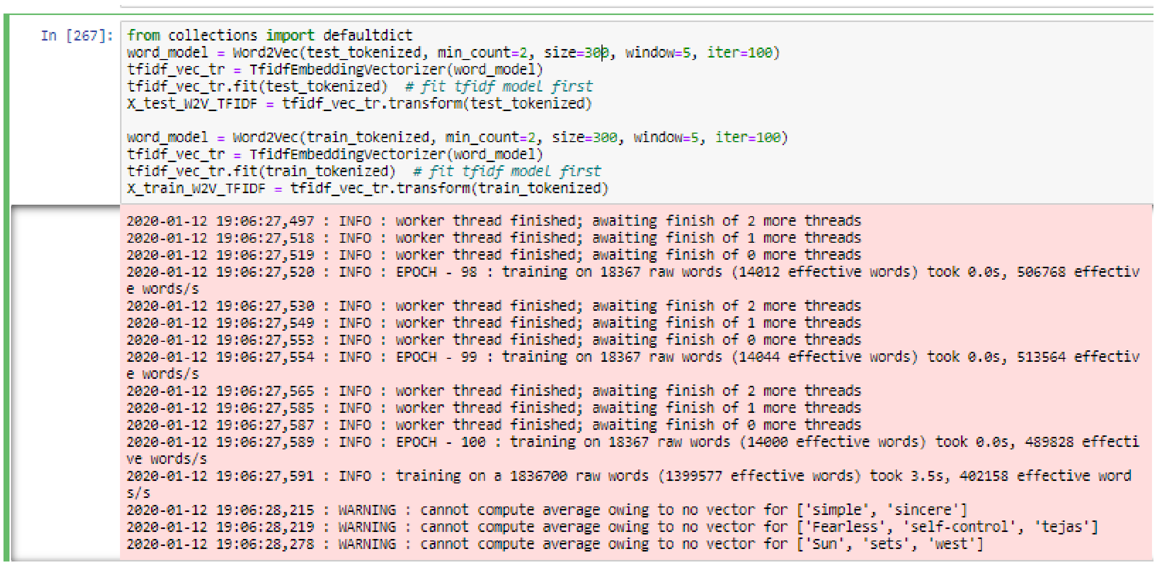
Higher weights are assigned to terms that are present frequently in a document and which are rare among all documents. On the other hand, a low score is assigned to terms, which are common across all documents.



```
In [266]: class TfidfEmbeddingVectorizer(object):
              def __init__(self, word_model):
                  self.word_model = word_model
                  self.word_idf_weight = None
                  self.vector_size = word_model.wv.vector_size
              def fit(self, docs):  # comply with scikit-learn transformer requirement
                  text_docs = []
                  for doc in docs:
                      text_docs.append(" ".join(doc))
                  tfidf = TfidfVectorizer()
                  tfidf.fit(text_docs)
                  max_idf = max(tfidf.idf_)  # used as default value for defaultdict
                  self.word_idf_weight = defaultdict(lambda: max_idf,
                                                     [(word, tfidf.idf_[i]) for word, i in tfidf.vocabulary_.items()])
                  return self
              def transform(self, docs):  # comply with scikit-learn transformer requirement
                  doc_word_vector = self.word_average_list(docs)
                  return doc_word_vector
              def word_average(self, sent):
                  mean = []
                  for word in sent:
                      if word in self.word_model.wv.vocab:
                          mean.append(self.word_model.wv.get_vector(word) * self.word_idf_weight[word])  # idf weighted
                  if not mean:  # empty words
                      # If a text is empty, return a vector of zeros.
                      logging.warning("cannot compute average owing to no vector for {}".format(sent))
                      return np.zeros(self.vector_size)
                  else:
                      mean = np.array(mean).mean(axis=0)
                      return mean
              def word_average_list(self, docs):
                  return np.vstack([self.word_average(sent) for sent in docs])
```

Figure 22 TF-IDF Embedding Vectorizer class

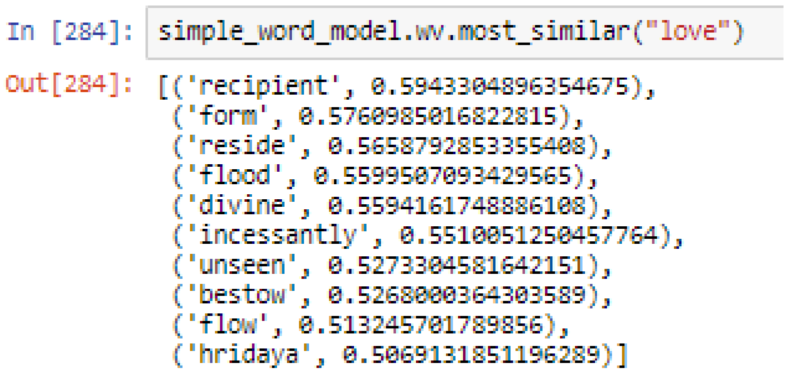
TF-IDF embedding vectorizer is created after 100 epochs. The resultant word model is of 300 dimensions trained on GoogleNews-vectors-negative300.bin.gz corpus. After training on a 1836700 raw words (1399577 effective words) took 3.5s, 402158 effective words/s 9 words average could not be computed and we ignore the warning.



```
In [267]: from collections import defaultdict
          word_model = Word2Vec(test_tokenized, min_count=2, size=300, window=5, iter=100)
          tfidf_vec_tr = TfidfEmbeddingVectorizer(word_model)
          tfidf_vec_tr.fit(test_tokenized)  # fit tfidf model first
          X_test_W2V_TFIDF = tfidf_vec_tr.transform(test_tokenized)

          word_model = Word2Vec(train_tokenized, min_count=2, size=300, window=5, iter=100)
          tfidf_vec_tr = TfidfEmbeddingVectorizer(word_model)
          tfidf_vec_tr.fit(train_tokenized)  # fit tfidf model first
          X_train_W2V_TFIDF = tfidf_vec_tr.transform(train_tokenized)

          2020-01-12 19:06:27,497 : INFO : worker thread finished; awaiting finish of 2 more threads
          2020-01-12 19:06:27,518 : INFO : worker thread finished; awaiting finish of 1 more threads
          2020-01-12 19:06:27,519 : INFO : worker thread finished; awaiting finish of 0 more threads
          2020-01-12 19:06:27,520 : INFO : EPOCH - 98 : training on 18367 raw words (14012 effective words) took 0.0s, 506768 effectiv
          e words/s
          2020-01-12 19:06:27,530 : INFO : worker thread finished; awaiting finish of 2 more threads
          2020-01-12 19:06:27,549 : INFO : worker thread finished; awaiting finish of 1 more threads
          2020-01-12 19:06:27,553 : INFO : worker thread finished; awaiting finish of 0 more threads
          2020-01-12 19:06:27,554 : INFO : EPOCH - 99 : training on 18367 raw words (14044 effective words) took 0.0s, 513564 effectiv
          e words/s
          2020-01-12 19:06:27,565 : INFO : worker thread finished; awaiting finish of 2 more threads
          2020-01-12 19:06:27,585 : INFO : worker thread finished; awaiting finish of 1 more threads
          2020-01-12 19:06:27,587 : INFO : worker thread finished; awaiting finish of 0 more threads
          2020-01-12 19:06:27,589 : INFO : EPOCH - 100 : training on 18367 raw words (14000 effective words) took 0.0s, 489828 effecti
          ve words/s
          2020-01-12 19:06:27,591 : INFO : training on a 1836700 raw words (1399577 effective words) took 3.5s, 402158 effective word
          s/s
          2020-01-12 19:06:28,215 : WARNING : cannot compute average owing to no vector for ['simple', 'sincere']
          2020-01-12 19:06:28,219 : WARNING : cannot compute average owing to no vector for ['Fearless', 'self-control', 'tejas']
          2020-01-12 19:06:28,278 : WARNING : cannot compute average owing to no vector for ['Sun', 'sets', 'west']
```

Figure 23 TF-IDF Embedding Vectorizer

Most similar word representation is analysed to get an overview of the Word2Vec embedding.

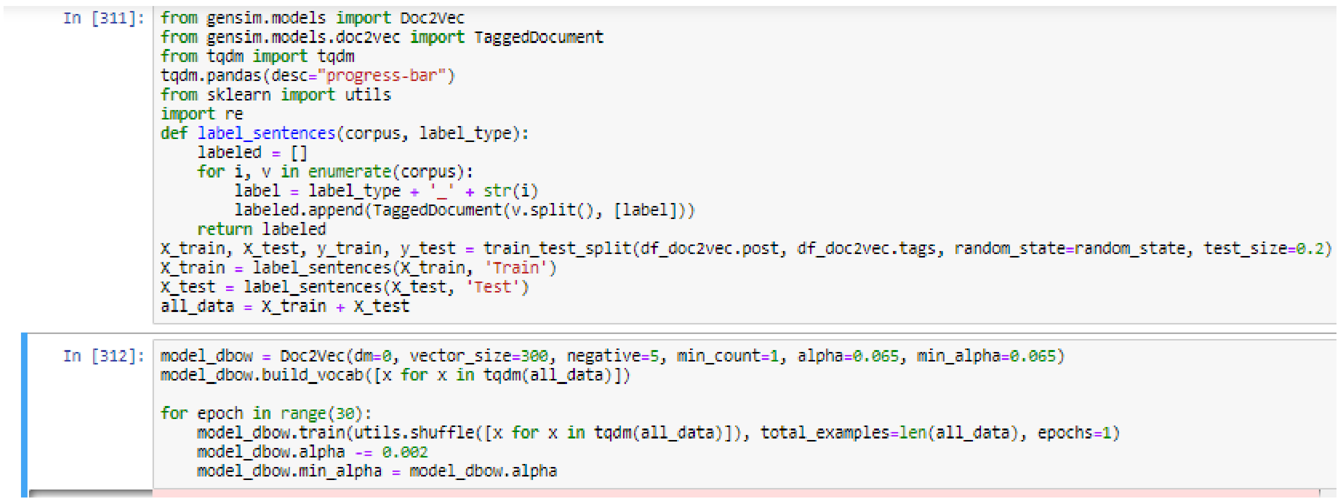


```
In [284]: simple_word_model.wv.most_similar("love")

Out[284]: [('recipient', 0.5943304896354675),
           ('form', 0.5760985016822815),
           ('reside', 0.5658792853355408),
           ('flood', 0.5599507093429565),
           ('divine', 0.559416174886108),
           ('incessantly', 0.5510051250457764),
           ('unseen', 0.5273304581642151),
           ('bestow', 0.5268000364303589),
           ('flow', 0.513245701789856),
           ('hridaya', 0.5069131851196289)]
```

Figure 24 Similar Words representation

3.3.5 Paragraph Embeddings

Doc2Vec is a paragraph and document embedding technique via distributed memory and distributed bag of words model [15] [20]. The algorithms use either hierarchical softmax / negative sampling [17] [18]. The model is similar to Word2Vec but introduces a tag. The tag is a word that is in every context of the document. ‘Distributed memory’ (PV-DM) is used is used to build vocabulary from a sequence of documents

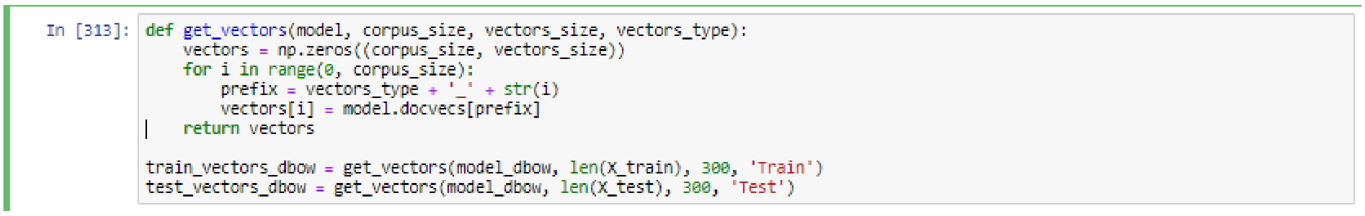


```
In [311]: from gensim.models import Doc2Vec
          from gensim.models.doc2vec import TaggedDocument
          from tqdm import tqdm
          tqdm.pandas(desc="progress-bar")
          from sklearn import utils
          import re
          def label_sentences(corpus, label_type):
              labeled = []
              for i, v in enumerate(corpus):
                  label = label_type + '_' + str(i)
                  labeled.append(TaggedDocument(v.split(), [label]))
              return labeled
          X_train, X_test, y_train, y_test = train_test_split(df_doc2vec.post, df_doc2vec.tags, random_state=random_state, test_size=0.2)
          X_train = label_sentences(X_train, 'Train')
          X_test = label_sentences(X_test, 'Test')
          all_data = X_train + X_test
```

```
In [312]: model_dbow = Doc2Vec(dm=0, vector_size=300, negative=5, min_count=1, alpha=0.065, min_alpha=0.065)
          model_dbow.build_vocab([x for x in tqdm(all_data)])

          for epoch in range(30):
              model_dbow.train(utils.shuffle([x for x in tqdm(all_data)]), total_examples=len(all_data), epochs=1)
              model_dbow.alpha -= 0.002
              model_dbow.min_alpha = model_dbow.alpha
```

Figure 25 Doc2Vec Tagging

Gensim's Doc2Vec implementation requires each document or paragraph to have a label associated with it. This is performed using the TaggedDocument method. The format will be "TRAIN_i" or "TEST_i" where "i" is a dummy index of the post. From the trained Doc2Vec model a vector is returned and the function parameters are the trained Doc2Vec model, itself Data Size, Size of embedding vector {300 dimensions in this case}, Training / Testing data.

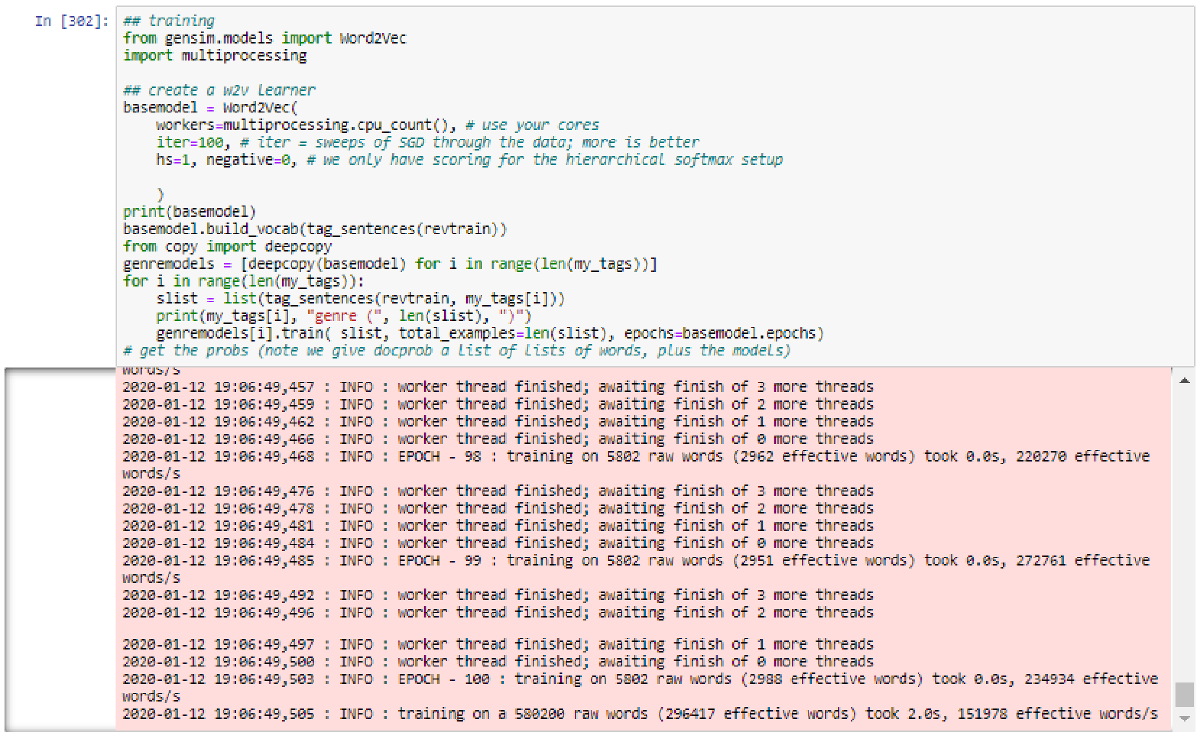


```
In [313]: def get_vectors(model, corpus_size, vectors_size, vectors_type):
              vectors = np.zeros((corpus_size, vectors_size))
              for i in range(0, corpus_size):
                  prefix = vectors_type + '_' + str(i)
                  vectors[i] = model.docvecs[prefix]
              return vectors

          train_vectors_dbow = get_vectors(model_dbow, len(X_train), 300, 'Train')
          test_vectors_dbow = get_vectors(model_dbow, len(X_test), 300, 'Test')
```

Figure 26 300 Dimensions Doc2Vec

3.3.6 Deep IR

Deep Inverse Regression [15] is implemented genism word2vec and Bayesian inversion. The implementation is simple where document probability for tags is obtained using base word2vec model of Gensim.

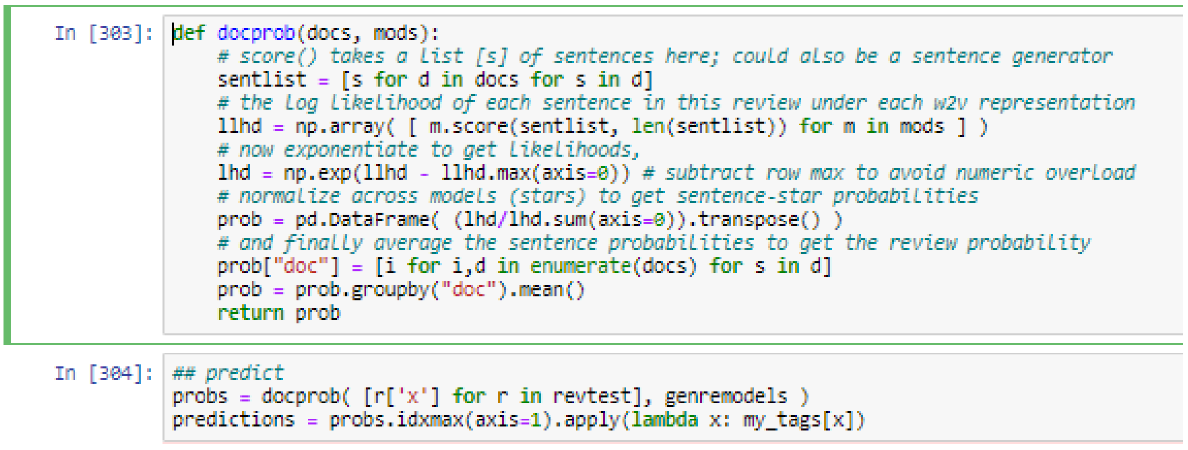


```
In [302]: ## training
          from gensim.models import Word2Vec
          import multiprocessing

          ## create a w2v learner
          basemodel = Word2Vec(
              workers=multiprocessing.cpu_count(), # use your cores
              iter=100, # iter = sweeps of SGD through the data; more is better
              hs=1, negative=0, # we only have scoring for the hierarchical softmax setup

              )
          print(basemodel)
          basemodel.build_vocab(tag_sentences(revtrain))
          from copy import deepcopy
          genremodels = [deepcopy(basemodel) for i in range(len(my_tags))]
          for i in range(len(my_tags)):
              slist = list(tag_sentences(revtrain, my_tags[i]))
              print(my_tags[i], "genre (", len(slist), ")")
              genremodels[i].train( slist, total_examples=len(slist), epochs=basemodel.epochs)
          # get the probs (note we give docprob a list of lists of words, plus the models)
```

```
words/s
2020-01-12 19:06:49,457 : INFO : worker thread finished; awaiting finish of 3 more threads
2020-01-12 19:06:49,459 : INFO : worker thread finished; awaiting finish of 2 more threads
2020-01-12 19:06:49,462 : INFO : worker thread finished; awaiting finish of 1 more threads
2020-01-12 19:06:49,466 : INFO : worker thread finished; awaiting finish of 0 more threads
2020-01-12 19:06:49,468 : INFO : EPOCH - 98 : training on 5802 raw words (2962 effective words) took 0.0s, 220270 effective
words/s
2020-01-12 19:06:49,476 : INFO : worker thread finished; awaiting finish of 3 more threads
2020-01-12 19:06:49,478 : INFO : worker thread finished; awaiting finish of 2 more threads
2020-01-12 19:06:49,481 : INFO : worker thread finished; awaiting finish of 1 more threads
2020-01-12 19:06:49,484 : INFO : worker thread finished; awaiting finish of 0 more threads
2020-01-12 19:06:49,485 : INFO : EPOCH - 99 : training on 5802 raw words (2951 effective words) took 0.0s, 272761 effective
words/s
2020-01-12 19:06:49,492 : INFO : worker thread finished; awaiting finish of 3 more threads
2020-01-12 19:06:49,496 : INFO : worker thread finished; awaiting finish of 2 more threads

2020-01-12 19:06:49,497 : INFO : worker thread finished; awaiting finish of 1 more threads
2020-01-12 19:06:49,500 : INFO : worker thread finished; awaiting finish of 0 more threads
2020-01-12 19:06:49,503 : INFO : EPOCH - 100 : training on 5802 raw words (2988 effective words) took 0.0s, 234934 effective
words/s
2020-01-12 19:06:49,505 : INFO : training on a 580200 raw words (296417 effective words) took 2.0s, 151978 effective words/s
```

Figure 27 DeepIR Implementation

The log likelihood of each sentence in this corpus under each word2Vec representation is obtained and exponentiated to get likelihoods. Then it is normalized across models to get sentence probabilities and finally average the sentence probabilities to get the label probability.



```
In [303]: def docprob(docs, mods):
              # score() takes a list [s] of sentences here; could also be a sentence generator
              sentlist = [s for d in docs for s in d]
              # the log likelihood of each sentence in this review under each w2v representation
              llhd = np.array( [ m.score(sentlist, len(sentlist)) for m in mods ] )
              # now exponentiate to get likelihoods,
              lhd = np.exp(llhd - llhd.max(axis=0)) # subtract row max to avoid numeric overload
              # normalize across models (stars) to get sentence-star probabilities
              prob = pd.DataFrame( (lhd/lhd.sum(axis=0)).transpose() )
              # and finally average the sentence probabilities to get the review probability
              prob["doc"] = [i for i,d in enumerate(docs) for s in d]
              prob = prob.groupby("doc").mean()
              return prob

In [304]: ## predict
          probs = docprob( [r['x'] for r in revtest], genremodels )
          predictions = probs.idxmax(axis=1).apply(lambda x: my_tags[x])
```

Figure 28 Training Deep_IR Model

The label prediction using Deep IR returns with accuracy of 68% and out of sample probability of only one label being corrected. This technique is dropped at his stage without further exploration as probability of 68% is not optimal.

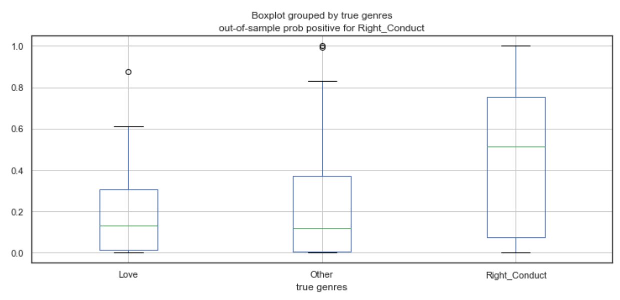


Figure 29 Confusion Matrix

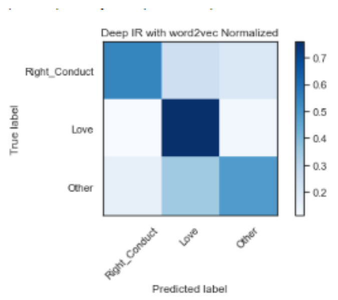


Figure 30 Predictions of Deep IR

3.4 Dimensionality Reduction

As the word embeddings consisting of number of features is large, 300 dimensions, these would be require intensive computing capacity. Principal Component Analysis and UMAP [13] techniques are used towards dimensionality reduction.

3.4.1 Principal Component Analysis

Principal component analysis techninque is used as a dimensionality reduction technique. PCA calculates the covariance matrix of the data points, calculates eigen vectors and respective eigen values, the first k eigen vectors based on descending order are the recommended dimension.

```
In [346]: plt.bar(range(1,301), var_explained, alpha=0.5, align='center', label='individual explained variance')
          plt.step(range(1,301),cum_var_exp, where= 'mid', label='cumulative explained variance')
          plt.ylabel('Explained variance ratio')
          plt.xlabel('Principal components')
          plt.legend(loc = 'best')
          plt.show()
```

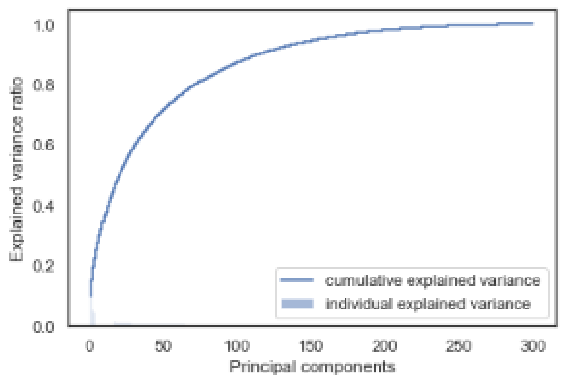
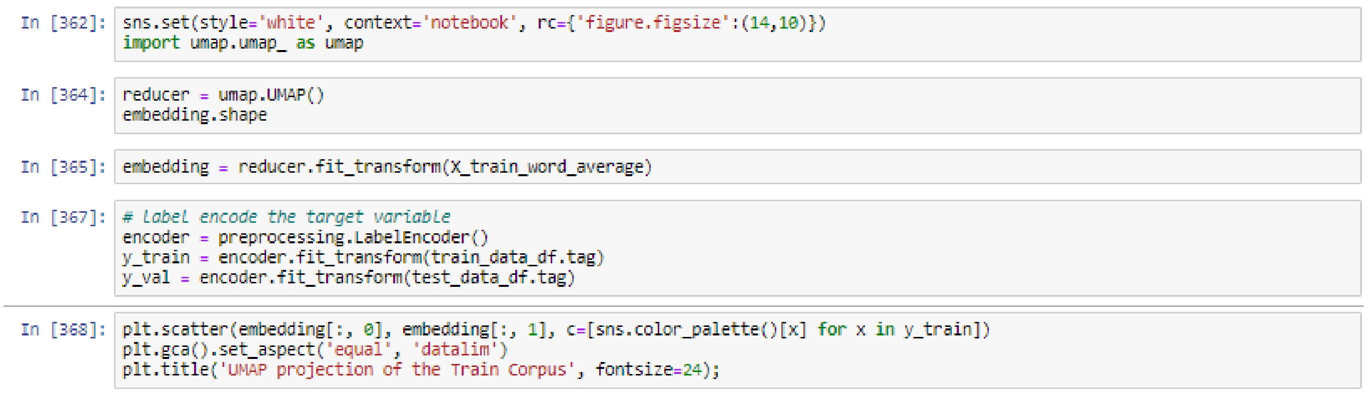


Figure 31 Principal Component Analysis Plot

Of 300 dimensional matrix, from above we plot we can clealry observe that 134 dimensions are able to explain 95% variance of data. Similarly PCA was applied on optimal datasets explained in section modelling.

3.4.2 Uniform Manifold Approximation and Projection

Uniform Manifold Approximation and Projection also known as UMAP [13] is used for general purpose manifold learning and dimensionality reduction. The embedding is found by searching for a low dimensional projection of the data that has the closest possible equivalent fuzzy topological structure. UMAP reduces many dimensions into two dimensions.



```
In [362]: sns.set(style='white', context='notebook', rc={'figure.figsize':(14,10)})
          import umap.umap_ as umap

In [364]: reducer = umap.UMAP()
          embedding.shape

In [365]: embedding = reducer.fit_transform(X_train_word_average)

In [367]: # Label encode the target variable
          encoder = preprocessing.LabelEncoder()
          y_train = encoder.fit_transform(train_data_df.tag)
          y_val = encoder.fit_transform(test_data_df.tag)

In [368]: plt.scatter(embedding[:, 0], embedding[:, 1], c=[sns.color_palette()[x] for x in y_train])
          plt.gca().set_aspect('equal', 'datalim')
          plt.title('UMAP projection of the Train Corpus', fontsize=24);
```

Figure 32 UMAP implementation

UMAP object is constructed and implemented to transform Train and Test data into two dimensions. From the visualization the data is scattered which can be easily visualized based on coloring based on label. While Label Other is seperable to certain extent, Love and Right Conduct seem to have heavy coorelation.

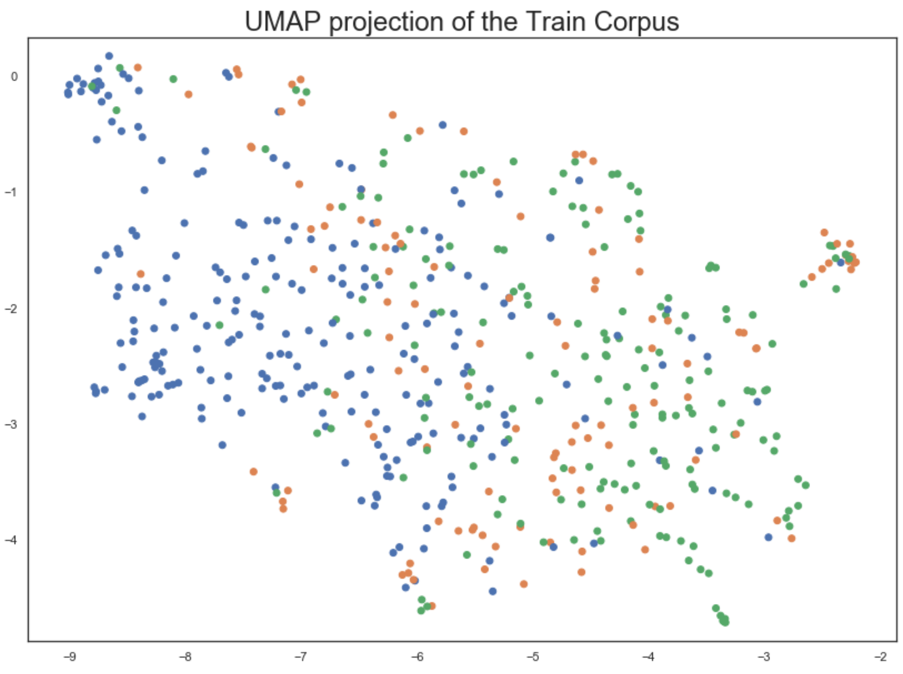


Figure 33 UMAP project of Train Data

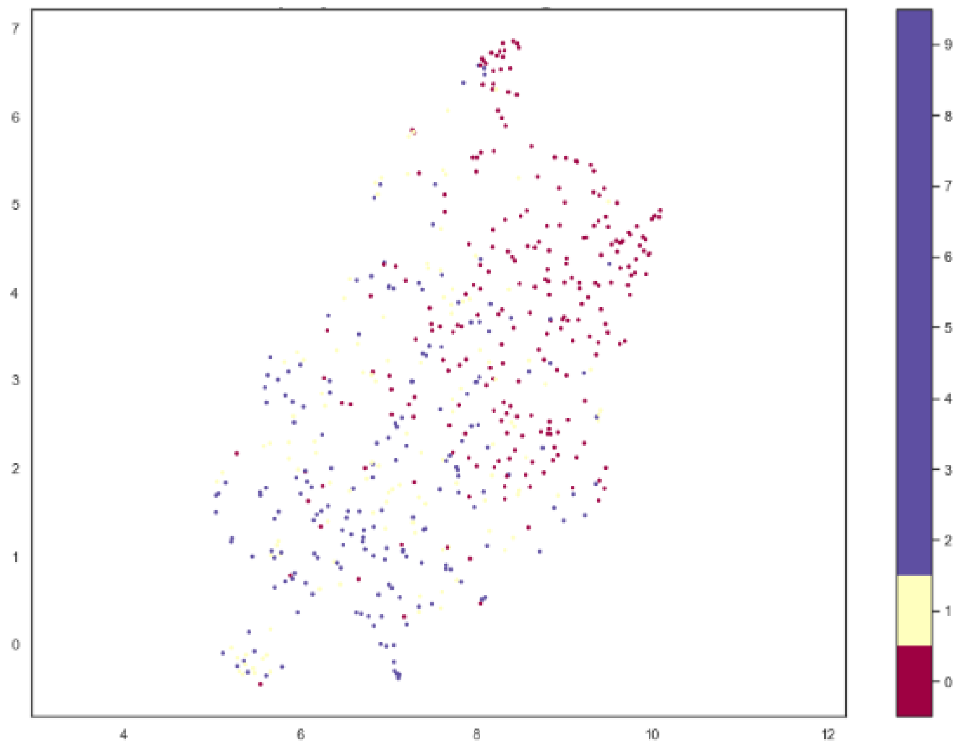


Figure 34 UMAP project of Test data

3.5 Modelling

Various Machine learning algorithms are used on the several train and test data sets discussed in section 3.4.2-Feature Extraction.

- Naive Bayes - Gaussian
- Naive Bayes - Multinomial
- Logistic regression
- SVM linear
- SVM rbf
- Random Forest
- Extreme Gradient Boosting
- k-nearest neighbors
- Rocchio classification
- Gradient Boosting Classifier
- Meta Bagging Decision Trees
- Meta Bagging-K Nearest
- Ada Boost Classifier

[21] AutoML is a function in H2O that automates the process of building a large number of models, with the goal of finding the "best" model without any prior knowledge or effort by the Data Scientist. 20 base models. AutoML trains and cross-validates the following algorithms (in the following order): three pre-specified XGBoost GBM (Gradient Boosting Machine) models, a fixed grid of GLMs, a default Random Forest (DRF), five pre-specified

H2O GBMs, a near-default Deep Neural Net, an Extremely Randomized Forest (XRT), a random grid of XGBoost GBMs, a random grid of H2O GBMs, and a random grid of Deep Neural Nets.

- DeepLearning (Fully-connected multi-layer artificial neural network)
- Extremely Randomized Trees
- Generalized Linear Model
- Random Forest and Extremely Randomized Trees
- Stacked Ensemble
- XGBoost GBM

The number of data sets the above algorithms are trained and tested are as follows. These are detailedly covered above.

- Bag of Words
- TF-IDF – Word Embedding
- TF-IDF – n gram Embedding
- TF-IDF – Character Embedding
- Pre-trained word embedding vector
- Word2Vec TF-IDF embedding vectorizer
- Word2Vec CBOW Word Averaging
- Doc2Vec Paragraph Embeddings 100 Dimensions
- Doc2Vec Paragraph Embeddings 300 Dimensions

Evaluation

Each classification model is evaluated and compared with other models. Accuracy, F1-scre are used as metrics. Though Accuracy { Number of predictions made correctly} metric is important, for this reseach F1-Score is considered as evaluating metric

- F1-score = (2 * Precision * Recall) / (Precision + Recall)

To obtain F1-score, precision and recall have been calculated however these are not discussed in this paper.

Confusion matrix to visually identify the accuracy for each class in a multi-class classification problem and Precision & Recall are referred to .

- Precision score quantifies the number of correct positive predictions made
  Precision = # True Positives / (# True Positives + # False Positives)
- Recall score quantifies the number of correct positive predictions made out of all positive predictions that could have been made
  Recall = # True Positives / (# True Positives + # False Negatives)

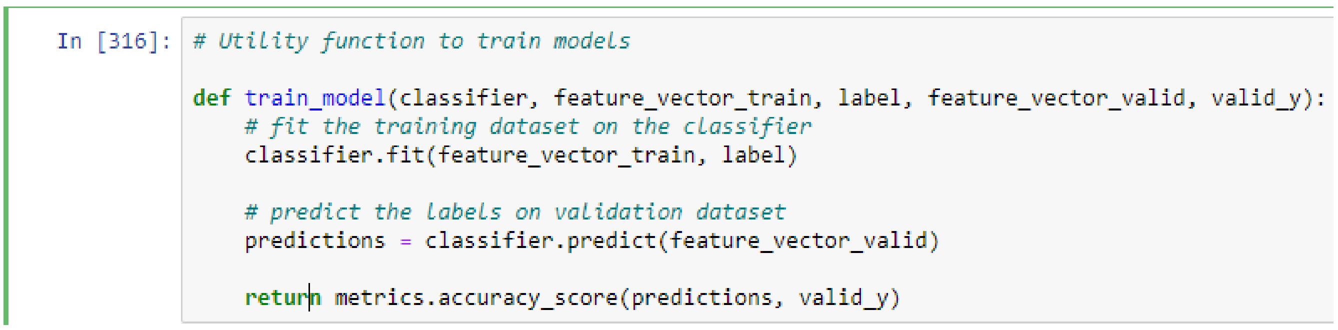
AUC - The area under the curve is the area under the receiver operating characteristic curve. The AUC is also calculated for each class, and an overall AUC is found by averaging.

CHAPTER 4

RESULTS AND EVALUATION

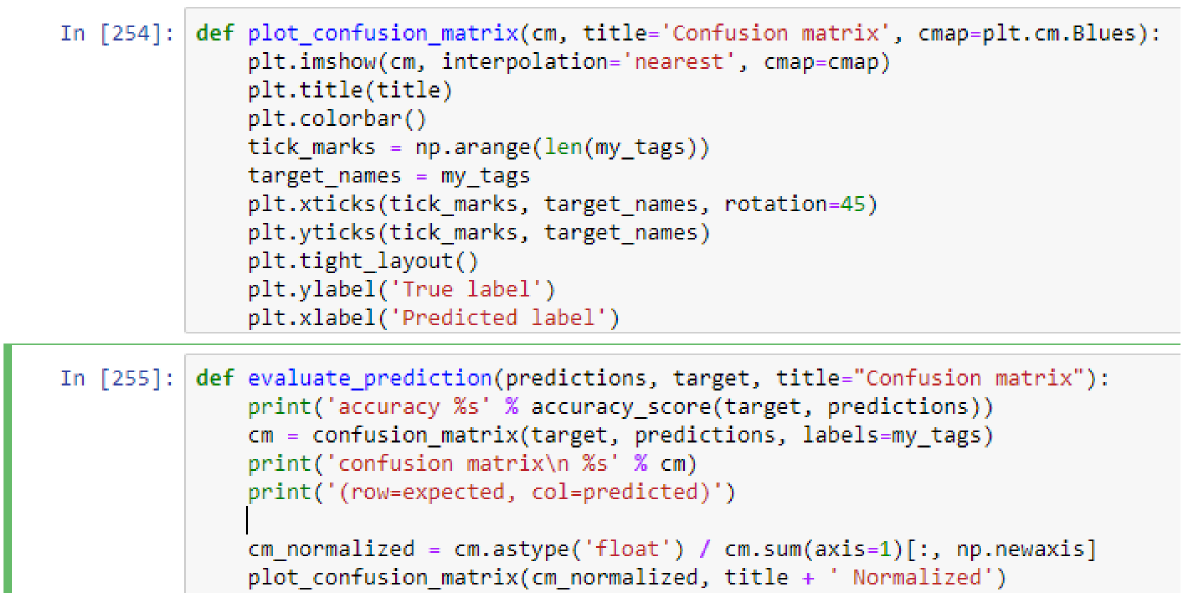
4.1 Introduction

Utility functions to train model, predict the labels on validation dataset and populate F1-score is created in Python. Similar function for confusion matrix and accuracy predictions is created to help as supporting data during modelling. In this paper, F1-score is presented in tabular format.



```
In [316]: # Utility function to train models

          def train_model(classifier, feature_vector_train, label, feature_vector_valid, valid_y):
              # fit the training dataset on the classifier
              classifier.fit(feature_vector_train, label)

              # predict the labels on validation dataset
              predictions = classifier.predict(feature_vector_valid)

              return metrics.accuracy_score(predictions, valid_y)
```

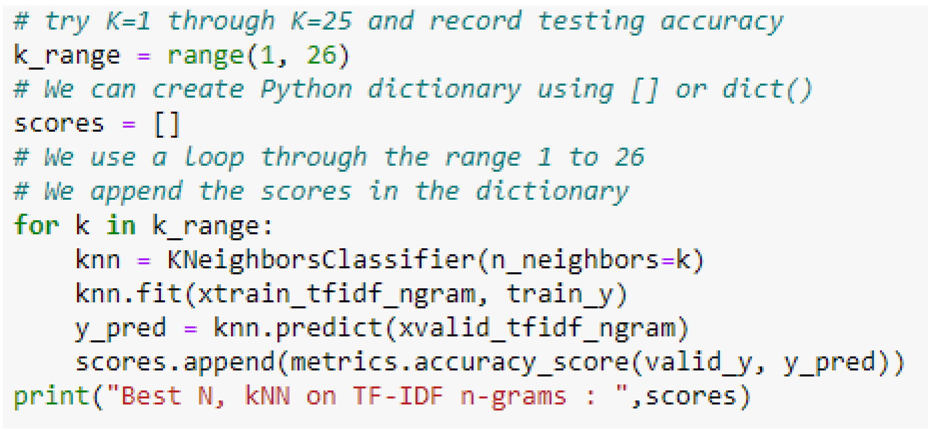
Figure 35 Utility function to Train Models



```
In [254]: def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
              plt.imshow(cm, interpolation='nearest', cmap=cmap)
              plt.title(title)
              plt.colorbar()
              tick_marks = np.arange(len(my_tags))
              target_names = my_tags
              plt.xticks(tick_marks, target_names, rotation=45)
              plt.yticks(tick_marks, target_names)
              plt.tight_layout()
              plt.ylabel('True label')
              plt.xlabel('Predicted label')

In [255]: def evaluate_prediction(predictions, target, title="Confusion matrix"):
              print('accuracy %s' % accuracy_score(target, predictions))
              cm = confusion_matrix(target, predictions, labels=my_tags)
              print('confusion matrix\n %s' % cm)
              print('(row=expected, col=predicted)')

              cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
              plot_confusion_matrix(cm_normalized, title + ' Normalized')
```

Figure 36 Utility Functions Confusion Matrix

To identify the best k value in k-nearest neighbors the classifier was run through a for loop where k is equal to 1 to 25.



```
# try K=1 through K=25 and record testing accuracy
k_range = range(1, 26)
# We can create Python dictionary using [] or dict()
scores = []
# We use a loop through the range 1 to 26
# We append the scores in the dictionary
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(xtrain_tfidf_ngram, train_y)
    y_pred = knn.predict(xvalid_tfidf_ngram)
    scores.append(metrics.accuracy_score(valid_y, y_pred))
print("Best N, kNN on TF-IDF n-grams : ",scores)
```

Figure 37 Finding optimum k

4.2 Model Validation

Utility functions to train model, predict the labels on validation dataset and populate F1-score is created in Python. Similar function for confusion matrix and accuracy predictions is created to help as supporting data during modelling. In this paper, F1-score is presented.

Data sets through stemming, lemmatization are trained and tested using below algorithms.

Table 1 ML results with Stemming & NLP Data Sets

f1-score (**Stemming**)	Bag of Words	TF-IDF – Word Embedding	TF-IDF – n gram Embedding	TF-IDF – Character Embedding	Pre-trained word embedding vector	Word2 Vec TF-IDF embedding vectorizer	Word2 Vec CBOW Word Averaging	Doc2Vec Paragraph Embeddings 100 Dimensions	Doc2Vec Paragraph Embeddings 300 Dimensions
Naive Bayes - Gaussian						41%	63%	49%	60%
Naive Bayes - MultinomialNB	67%	57%	60%	50%	33%				
Logistic regression	74%	74%	64%	76%	40%	41%	70%	61%	63%
SVM linear	48%	43%	43%	43%	49%	60%	39%	67%	64%
SVM rbf	57%	44%	44%	44%	45%	60%	39%	67%	64%
Random Forest	80%	69%	64%	73%	57%	58%	70%	64%	66%
Extreme Gradient Boosting	78%	83%	64%	81%	57%	49%	68%	65%	67%
k-nearest neighbors	62%	66%	60%	65%	51%	46%	58%	61%	62%

Rocchio classification	53%	71%	69%	69%	46%	42%	63%	49%	66%
Gradient Boosting Classifier	82%	80%	66%	81%	48%	58%	67%	64%	64%
Meta Bagging Decision Trees	84%	84%	66%	82%	44%	53%	51%	52%	61%
Meta Bagging-K Nearest	61%	66%	54%	54%	47%				
Ada Boost Classifier	74%	74%	67%	54%	50%	61%	41%	62%	64%

Table 2 ML results with Lemmatization & NLP Data Sets

f1-score (**Lemmatization**)	Bag of Words	TF-IDF – Word Embedding	TF-IDF – n gram Embedding	TF-IDF – Character Embedding	Pre-trained word embedding vector	Word2Vec TF-IDF embedding vectorizer	Word2Vec CBOW Word Averaging	Doc2Vec Paragraph Embeddings 100 Dimensions	Doc2Vec Paragraph Embeddings 300 Dimensions
Naive Bayes - Gaussian						63%	68%	57%	59%
Naive Bayes - Multinomial NB	75%	47%	56%	44%	33%				
Logistic regression	78%	78%	64%	75%	39%	44%	77%	62%	65%
SVM linear	49%	42%	42%	42%	44%	53%	47%	62%	64%
SVM rbf	49%	42%	42%	42%	44%	53%	47%	62%	64%
Random Forest	77%	75%	66%	80%	53%	49%	76%	66%	66%
Extreme Gradient Boosting	81%	75%	58%	83%	39%	49%	84%	63%	65%
k-nearest neighbors	64%	63%	64%	60%	44%	47%	65%	60%	66%
Rocchio classification	51%	73%	67%	71%	45%	44%	70%	46%	66%
Gradient Boosting Classifier	76%	80%	58%	83%	45%	49%	82%	60%	60%
Meta Bagging	81%	80%	73%	81%	44%	24%	74%	57%	53%

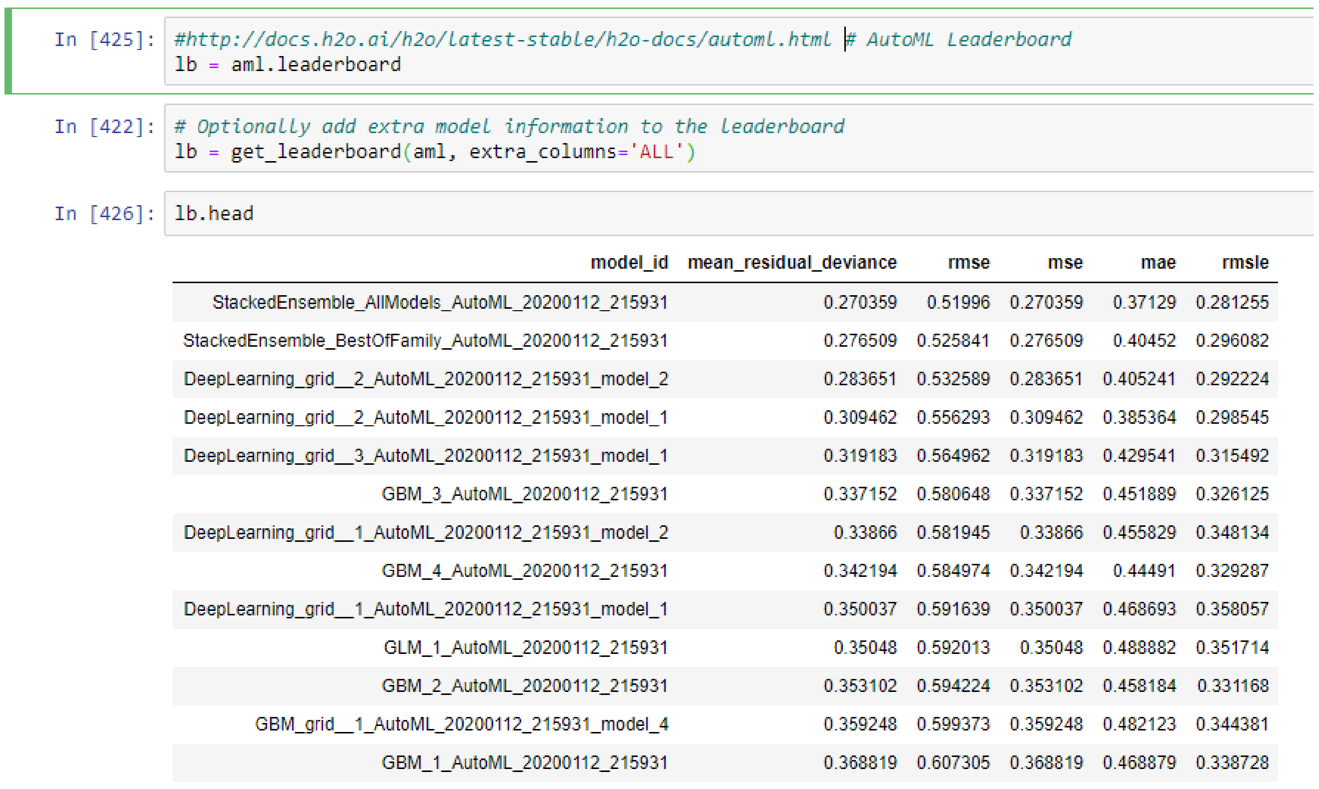
Decision Trees									
Meta Bagging-K Nearest	49%	60%	45%	61%	38%				
Ada Boost Classifier	70%	40%	67%	78%	44%	66%	44%	62%	58%
Soft Voting Classifier - (DecisionTreeClassifier, RandomForestClassifier, GradientBoostingClassifier, XGBClassifier)							69%		

Dimensionality reduction technique PCA and UMAP were applied on Word2Vec CBOW Word Averaging train and test data. The results were far from encouraging.

Table 3 Lemmatization CBOW ML results - Dimensionality Reduction

f1-score (Lemmatization) Word2Vec CBOW Word Averaging	PCA	UMAP
Naive Bayes - Gaussian	40.00%	32.59%
Naive Bayes - MultinomialNB	40.00%	27.41%
SVM linear	40.00%	32.59%
SVM rbf	40.00%	32.59%
Random Forest	25.19%	32.59%
Extreme Gradient Boosting	42.96%	29.63%
k-nearest neighbors	53.33%	39.26%
Rocchio classification	62.96%	32.59%
Gradient Boosting Classifier	34.07%	58.52%
Meta Bagging Decision Trees	34.81%	32.59%
Ada Boost Classifier	48.89%	60.74%

The results are clear from Machine Learning algorithms specified above. H20 Auto ML is used to look at Deep learning grids which are fully-connected multi-layer artificial neural network.



```
In [425]: #http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html # AutoML Leaderboard
          lb = aml.leaderboard
```

```
In [422]: # Optionally add extra model information to the leaderboard
          lb = get_leaderboard(aml, extra_columns='ALL')
```

```
In [426]: lb.head
```

model_id	mean_residual_deviance	rmse	mse	mae	rmsle
StackedEnsemble_AllModels_AutoML_20200112_215931	0.270359	0.51996	0.270359	0.37129	0.281255
StackedEnsemble_BestOfFamily_AutoML_20200112_215931	0.276509	0.525841	0.276509	0.40452	0.296082
DeepLearning_grid__2_AutoML_20200112_215931_model_2	0.283651	0.532589	0.283651	0.405241	0.292224
DeepLearning_grid__2_AutoML_20200112_215931_model_1	0.309462	0.556293	0.309462	0.385364	0.298545
DeepLearning_grid__3_AutoML_20200112_215931_model_1	0.319183	0.564962	0.319183	0.429541	0.315492
GBM_3_AutoML_20200112_215931	0.337152	0.580648	0.337152	0.451889	0.326125
DeepLearning_grid__1_AutoML_20200112_215931_model_2	0.33866	0.581945	0.33866	0.455829	0.348134
GBM_4_AutoML_20200112_215931	0.342194	0.584974	0.342194	0.44491	0.329287
DeepLearning_grid__1_AutoML_20200112_215931_model_1	0.350037	0.591639	0.350037	0.468693	0.358057
GLM_1_AutoML_20200112_215931	0.35048	0.592013	0.35048	0.488882	0.351714
GBM_2_AutoML_20200112_215931	0.353102	0.594224	0.353102	0.458184	0.331168
GBM_grid__1_AutoML_20200112_215931_model_4	0.359248	0.599373	0.359248	0.482123	0.344381
GBM_1_AutoML_20200112_215931	0.368819	0.607305	0.368819	0.468879	0.338728

Figure 38 H20's Auto ML

4.2 Interpretation of results

After viewing the results, the threshold of f1-score greater than 80% is considered. The optimum results are from Extreme Gradient Boosting, Gradient Boosting Classifier and Meta Bagging Decision Trees algorithms.

Word2Vec CBOW Word Averaging NLP technique with machine learning algorithm Extreme Gradient Boosting has slightly better results when Lemmatization word form normalization technique is used.

TF-IDF – Character Embedding NLP technique produced optimum results with machine learning algorithms Extreme Gradient Boosting, Gradient Boosting Classifier and Meta Bagging Decision Trees with both Stemming and Lemmatization word form normalization technique.

Bag of Words NLP technique produced optimum results with machine learning algorithms Meta Bagging Decision Trees and Gradient Boosting Classifier when Stemming word Form Normalization technique is used.

Table 4 Optimum Results

Word Form Normalization	NLP Technique	Machin Learning	f1-score
Lemmatization	Word2Vec CBOW Word Averaging	Extreme Gradient Boosting	84.33%
Stemming	Bag of Words	Meta Bagging Decision Trees	84.26%
Stemming	TF-IDF – Word Embedding	Meta Bagging Decision Trees	84.26%
Stemming	TF-IDF – Word Embedding	Extreme Gradient Boosting	83.33%
Lemmatization	TF-IDF – Character Embedding	Extreme Gradient Boosting	83.33%
Lemmatization	TF-IDF – Character Embedding	Gradient Boosting Classifier	83.33%
Stemming	Bag of Words	Gradient Boosting Classifier	82.41%
Stemming	TF-IDF – Character Embedding	Meta Bagging Decision Trees	82.41%
Lemmatization	Word2Vec CBOW Word Averaging	Gradient Boosting Classifier	81.69%
Stemming	TF-IDF – Character Embedding	Extreme Gradient Boosting	81.48%
Lemmatization	TF-IDF – Character Embedding	Meta Bagging Decision Trees	81.48%
Stemming	TF-IDF – Character Embedding	Gradient Boosting Classifier	80.56%
Lemmatization	Bag of Words	Extreme Gradient Boosting	80.56%
Lemmatization	Bag of Words	Meta Bagging Decision Trees	80.56%

4.3 Summary

Extreme Gradient Boosting and Meta Bagging Decision Trees have produced better results with the data sets of Word2Vec CBOW, Bag of words and TF-IDF word level embeddings.

The optimal process flow from cleaning data to modelling is proposed to conclude the research.

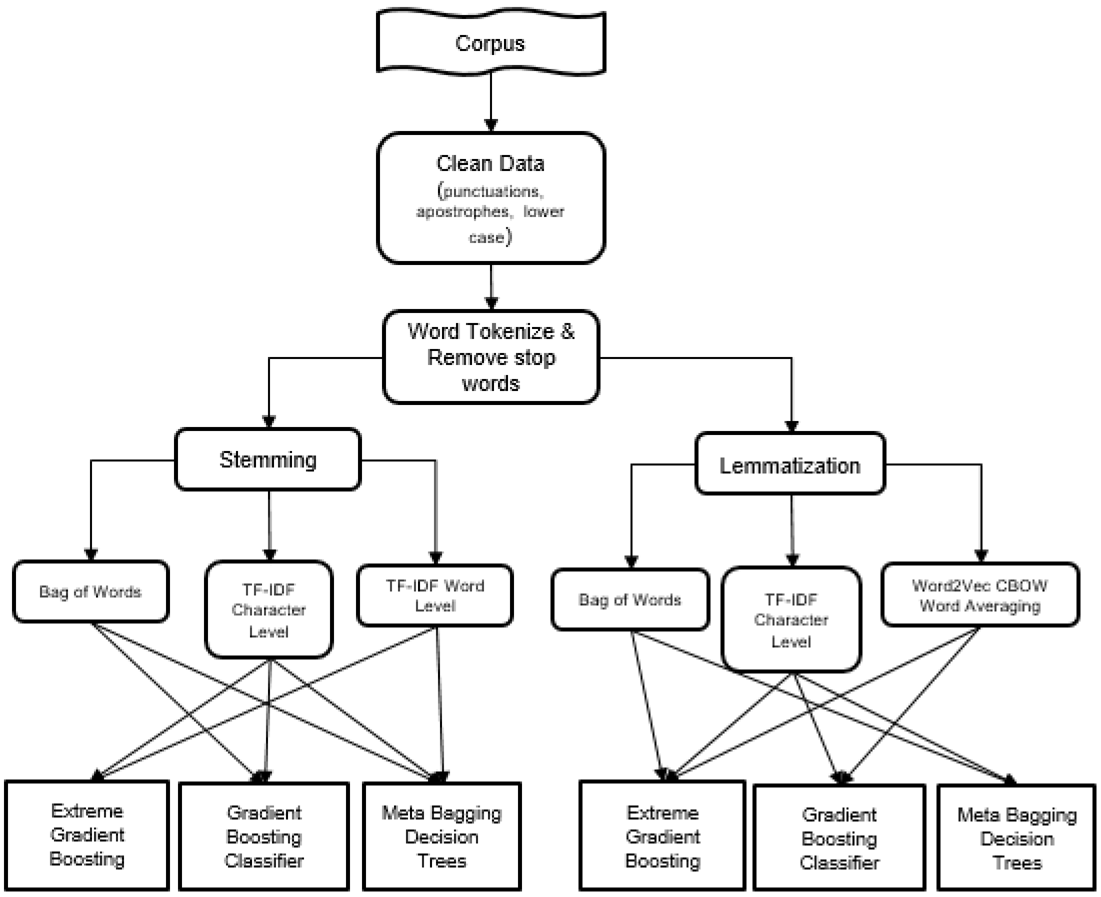

Corpus
Clean Data (punctuations, apostrophes, lower case)
Word Tokenize & Remove stop words
Stemming
Lemmatization
Bag of Words
TF-IDF Character Level
TF-IDF Word Level
Bag of Words
TF-IDF Character Level
Word2Vec CBOW Word Averaging
Extreme Gradient Boosting
Gradient Boosting Classifier
Meta Bagging Decision Trees
Extreme Gradient Boosting
Gradient Boosting Classifier
Meta Bagging Decision Trees

Figure 39 Optimal Process Flow

4.4 Future Scope for Improvement

This research was limited to classifying text across two human values. There is abundant scope to explore text classification under all of five human values as defined by Orator. Deep learning techniques, Bidirectional Encoder Representations from Transformers (BERT) can be explored towards multi text classification of human values and further improve f1-score accuracy.

REFERENCES

[1] Kowsari, Kamran and Jafari Meimandi, Kiana and Heidarysafa, Mojtaba and Mendu, Sanjana and Barnes, Laura E. and Brown, Donald E. (2019) Text Classification Algorithms: A Survey, *Special Issue "Machine Learning on Scientific Data and Information" 23 April 2019*

Available at: https://www.mdpi.com/2078-2489/10/4/150

[Accessed: 12TH November 2019]

[2] Xiang Zhang and Yann LeCun (2017) Which Encoding is the Best for Text Classification in Chinese, English, Japanese and Korean? [Online]

Available at: https://arxiv.org/pdf/1708.02657v2.pdf

[Accessed: 12TH November 2019]

[3] Shalendra Chhabra, William S. Yerazunis and Christian Siefkes (2016) Spam Filtering using a Markov Random Field Model with Variable Weighting Schemas *In Proc. 30th AAAI Conference on Artificial Intelligence*

Available at: https://arxiv.org/abs/1510.00098

[Accessed: 14TH November 2019]

[4] Kwan yi and Jamshid Beheshti (2009) A hidden Markov model-based text classification of medical documents *Journal of Information Science 35(1):67-81, February 2009*

Available at: https://www.researchgate.net/publication/220195924_A_hidden_Markov_model-based_text_classification_of_medical_documents

[Accessed: 14TH November 2019]

[5] Gongde Guo, Hui Wang, David A. Bell, Yaxin Bi and Kieran Greer (2006) Using kNN model for automatic text categorization [online]

Available at:
https://www.researchgate.net/publication/257432190_Using_kNN_model_for_automatic_text_categorization/citations

[Accessed: 25TH November 2019]

[6] Jincy B.Chrystal and Stephy Joseph (2015) Multi-Label Classification of Product Reviews Using Structured SVM *International Journal of Artificial Intelligence & Applications (IJAIA) Vol. 6, No. 3, May 2015*
Available at:
https://www.researchgate.net/publication/279250081_Multi-Label_Classification_of_Product_Reviews_Using_Structured_SVM
[Accessed: 26TH November 2019]

[7] Tuomo Korenius, Jorma Laurikkala1, Kalervo Järvelin and Martti Juhola1 (2004) Stemming and Lemmatization in the Clustering of Finnish Text Documents *Department of Computer Sciences, Center for Advanced Studies, FIN-33014 University of Tampere, Finland*
Available at:
https://www.researchgate.net/publication/221615320_Stemming_and_lemmatization_in_the_clustering_of_Finnish_text_documents
[Accessed: 26TH November 2019]

[8] Pu Han, Si Shen, Dongbo Wang and Yanyun Liu (2012) The influence of word normalization in English document clustering *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*
Available at: https://ieeexplore.ieee.org/document/6272740/authors#authors
[Accessed: 26TH November 2019]

[9] Xiaojun Wan and Tianming Wang (2016) Automatic Labeling of Topic Models Using Text Summaries *Published in ACL 2016*
Available at: https://www.semanticscholar.org/paper/Automatic-Labeling-of-Topic-Models-Using-Text-Wan-Wang/4cc4898e86c4e9d2a87bbb26f18ec6ebb79e9ded
[Accessed: 26TH November 2019]

[10] Benjamin Fayyazuddin Ljungberg (2017) Dimensionality reduction for bag-of-words models: PCA vs LSA
Available at: http://cs229.stanford.edu/proj2017/final-reports/5163902.pdf
[Accessed: 26TH November 2019]

[11] Vikas Raunak (2017) Simple and Effective Dimensionality Reduction for Word Embeddings *accepted at NIPS 2017 LLD Workshop* arXiv:1708.03629v3 [cs.CL]
Available at: https://arxiv.org/abs/1708.03629
[Accessed: 26TH November 2019]

[12] Ziqiao Weng (2019) From Conventional Machine Learning to AutoML *April 2019 Journal of Physics Conference Series 1207(1):012015*
Available at:
https://www.researchgate.net/publication/332690245_From_Conventional_Machine_Learning_to_AutoML
[Accessed: 26TH November 2019]

[13] Leland McInnes, John Healy, Nathaniel Saul and Lukas Großberger (2018) UMAP: Uniform Manifold Approximation and Projection *September 2018 The Journal of Open Source Software 3(29):861*
Available at:
https://www.researchgate.net/publication/327391699_UMAP_Uniform_Manifold_Approximation_and_Projection
[Accessed: 10TH December 2019]

[14] Data Source – Free for use Sri Sathya Sai Books and Publications Trust, Prashanthi Nilayam
Available at: http://www.sssbpt.info/english/vsathyasai.htm

[15] Matt Taddy (2015) Document Classification by Inversion of Distributed Language Representation
Available at: https://arxiv.org/pdf/1504.07295v3.pdf
[Accessed: 10TH December 2019]

[16] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, A. Joulin. Advances in Pre-Training Distributed Word @inproceedings{mikolov2018advances, title={Advances in Pre-Training Distributed Word Representations}, author={Mikolov, Tomas and Grave, Edouard and Bojanowski, Piotr and Puhrsch, Christian and Joulin, Armand}, booktitle={Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)}, year={2018} }

Available at: https://fasttext.cc/docs/en/english-vectors.html

[Accessed: 10TH December 2019]

[17] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR, 2013.

Available at : https://code.google.com/archive/p/word2vec/

[Accessed: 8TH December 2019]

[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS, 2013.

Available at : https://code.google.com/archive/p/word2vec/

[Accessed: 8TH December 2019]

[19] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. In Proceedings of NAACL HLT, 2013.

Available at: https://code.google.com/archive/p/word2vec/

[Accessed: 8TH December 2019]

[20] Quoc Le and Tomas Mikolov (2014) Distributed Representations of Sentences and Documents , Google Inc

Available at: https://arxiv.org/pdf/1405.4053v2.pdf

[Accessed: 8TH December 2019]

[21] AutoML: Automatic Machine Learning – Online reference & study [Accessed: 20TH January 2020]

Available at: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html