Article Theme Classification

Group Name: Group 3

Group Members:

First name	Last Name	Student number
Arjun	Chowhan	C0850490
Harshad	Patil	C0852307
Niteesha	Balla	C0850488
Tejaswi	Kalla	C0852124
Yashwanth	Paladi	C0849467

Submission date: ***13th April 2022***

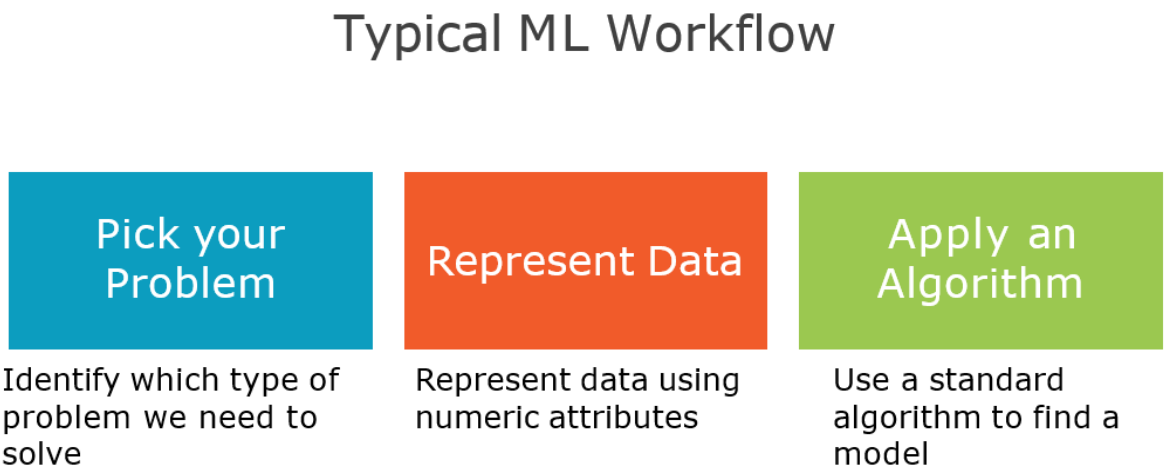
Contents

Abstract

Everyone wants their work done in a few seconds in today's busy schedule. There is not enough time to read the news or the articles ultimately to gain knowledge of the updates in the world. Many projects are proposed to address this problem to a certain extent, like the Audible App, which reads out the content in books, podcasts that we can listen to while on our way to work, and many more. To make it much easier to find out the exact content of their interest domain, we have developed a machine learning model trained on a corpus of articles and gives us the theme of an article when passed in as input. Users can get to know the piece's context without reading the full article. In this way, users can productively use their time without wasting time reading all the articles that are not useful for them.

Introduction

The theme of a story or an article is essential because it is a part of why the author wrote that story. The author has a message he wants to share with readers, and he uses his story as a way to get that message across. Sometimes people are interested in reading only the articles related to technology or specific to their domain. They might not have enough time to glance at every article and then figure out their interests in this busy world. Our model will help them classify the theme of articles and provide them with the best articles of their interest domain. Our objective is to classify the theme of an article using a classifier model and train it with the corpus of articles along with their themes dataset. Let us understand the machine learning workflow.



1. The first step is to pick the type of problem that we're trying to solve. We need to first identify which standard category of machine learning problems our problem falls into. Here we are given a large group of articles, and we need to divide these articles into clusters or groups on the basis of some common attributes. We need to classify the articles based on their themes. This is a classification problem.
2. The next step is to represent our text data using numeric attributes called features, and this process of extracting numeric attributes from text is called feature extraction. With the TF-IDF method, each document is now a tuple of n numbers where n is the total number of distinct words across all documents.
3. And finally, we would pick and apply a standard algorithm on that data. A tuple of n numbers can be represented as a point in an n-dimensional hypercube. So, we can use the K Nearest Neighbors classifier algorithm to classify the themes of the articles. Also, we can use the KMeans algorithm to define the clusters of the articles.

Below is the overview of the steps that we followed in our project:

1. **Building a corpus of Articles –**

   In order to train a model, we need to build a dataset that consists of articles and the theme of the article. For this, we have gathered a set of articles from a tech blog. For defining the themes of all the articles, we can label them manually by reading each article or we can use a clustering algorithm on the articles and assign a cluster to each article. In this project, we have used the clustering algorithm to figure out the themes of all the articles.

2. **Feature Extraction –**

   Our dataset consists of data in text format and as the machine learning model can accept only the data in numerical format, we need to convert our text data into numerical format. For this, we have used term frequency and inverse document frequency method (TF-IDF). This method creates a tuple of all the unique words from the corpus of articles called the bag of words model and then stores the TF-IDF values of all the articles in a tuple format.

3. **Identifying the article themes using K-Means Clustering algorithm –**

   After converting all the text data into numerical format, we need to train the K-Means clustering algorithm and form clusters. From the clusters, we can figure out the common theme of the cluster by finding the most frequent words of that cluster.

4. **Assigning a theme to an article –**

To assign a theme to any article, we need to train a classifier algorithm with the articles' corpus and their respective themes. For this, we chose the K Nearest Neighbors algorithm which assigns the theme based on the distance of the article point to the K nearest article points in an N-dimensional hypercube.

Methods

To accomplish the task which we have defined in the introduction, we used some built-in python libraries and defined some custom methods. In this section, we define the technical methods and approaches we used in our project.

1. Gather a corpus of articles from a given URL –

To gather the corpus of articles we need below built-in python libraries.

- **urllib and BeautifulSoup** –

```
# to download and parse a HTML webpage
import urllib
from bs4 import BeautifulSoup
```

- **urllib** –

Urllib package is the URL handling module for python. It is used to fetch URLs (Uniform Resource Locators). It uses the *urlopen* function and is able to fetch URLs using a variety of different protocols.

Urllib is a package that collects several modules for working with URLs, such as:

- urllib.request for opening and reading.
- urllib.parse for parsing URLs.
- urllib.error for the exceptions raised.
- urllib.robotparser for parsing robot.txt files.

- **BeautifulSoup –**

  There are mainly two ways to extract data from a website: Use the API of the website (if it exists). For example, Facebook has the Facebook Graph API which allows retrieval of data posted on Facebook. Access the HTML of the webpage and extract useful information/data from it. This technique is called web scraping or web harvesting or web data extraction. Beautiful Soup is a library that makes it easy to scrape information from web pages. It sits atop an HTML or XML parser, providing Pythonic idioms for iterating, searching, and modifying the parse tree.

Get all the article URLs from a blog URL –

Below is the URL of a blog where few technology related articles are summarized and posted every day.

blogUrl = "http://doxydonkey.blogspot.in"

In the blog URL, we have 7 articles and a link at last to retrieve the previous articles. Each URL will have the blog posts or tech news summaries for seven days.

There is a link on the homepage called Older Posts, which will give a URL with the summaries for the previous seven days.

The below function will get all the URL links of the previous articles and append them to the variable links.

These will be represented using a tag. soup.findAll will find all the links which are present on the homepage. We want to find the link which has the text called Older Posts. For each link, we can find the URL by using the attribute href and the text by using the attribute title. If the title is equal to Older Posts, then we want to actually keep that link. So we append it where we are storing all the links that we will later parse for articles.

The below method is customized only to this website and if we want to use it for another website we need to make some changes to the code by inspecting the HTML code of the specific website.

```
[ ]  #Gets all the urls from the homepage and append it to a variable
     def getAllBlogPosts(url,links):
         response = urllib.request.urlopen(url)
         soup = BeautifulSoup(response)
         for a in soup.findAll('a'):
             try:
                 url = a['href']
                 title = a['title']
                 if title == "Older Posts":
                     links.append(url)
                     getAllBlogPosts(url,links)
             except:
                 title = ""
         return
```

We create a list of links and call the above function by passing the blogURL and the list.

```
#Create a list and store all the article URLs
links = []
getAllBlogPosts(blogUrl,links)
```

The getBlogText function gets all the articles from the given URLs. Each URL contains 7 articles and appends all the article texts to the posts variable and returns it.

To understand how to parse the article text, we can go to the browser and inspect the article text element. Right-click on the article text and click Inspect, and this will open up a developer tools window which shows you the HTML corresponding to the element that we are looking at. Each article that we want to collect is a bullet point on this page, and bullet points are represented using the tag li. Each day's tech news summary is one post, which might contain multiple such bullet points. Each day's summary is contained within a div element whose class name is post-body. So in order to collect all the articles, we need to find divs which have this class, post-body, and within those divs, we need to find the bullet points represented by the li tag. Given a URL containing multiple tech news summaries, this function will collect all the articles from that blog page.

```
#get the article text from the URL
def getBlogText(testUrl):
    response = urllib.request.urlopen(testUrl)
    soup = BeautifulSoup(response)
    mydivs = soup.findAll("div", {"class":'post-body'})

    posts =[]
    for div in mydivs:
        posts += map(lambda p:p.text.encode('ascii', errors='replace').replace(b"?",b" "), div.findAll("li"))
    return posts
```

let's create a variable BlogPosts to store the corpus of articles. We can iterate over the links to extract the articles using the getBlogText function and store them in the variable BlogPosts.

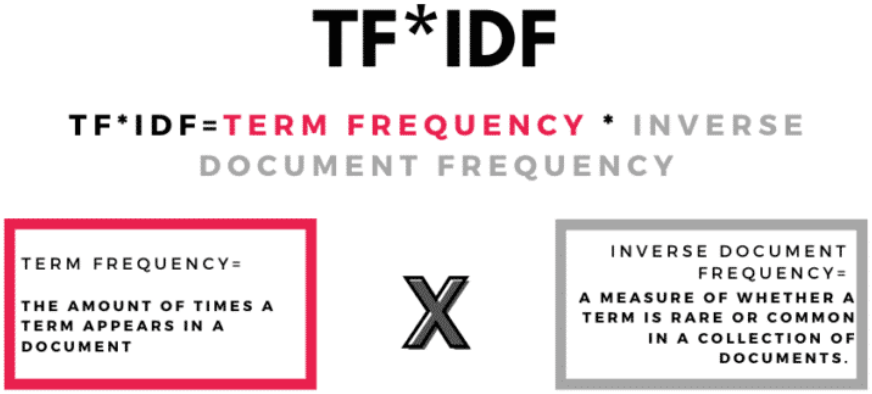
```
BlogPosts = []
for link in links:
    BlogPosts += getBlogText(link)
```

1. Feature Extraction - Extract features from all the articles –

The process of extracting numeric attributes from text is called feature extraction.

There are two methods, Term Frequency and TF-IDF.

Both the methods require a bag of words model means to create a list representing the universe of all words that can appear in any text from the corpus.



We first need to take all the articles and represent them using the TF-IDF representation.

Scikit-learn is a Python module with a lot of built-in functionality available for machine learning tasks, and this contains a feature extraction module, which allows you to perform TF-IDF representation.

TfidfVectorizer -

Convert a collection of raw documents to a matrix of TF-IDF features.

Equivalent to CountVectorizer followed by TfidfTransformer.

So we import the TfidfVectorizer from sklearn.feature_extraction.text, and we instantiate a vectorizer object.

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

We need to mention the stop words parameter to ignore the stop words from the articles corpus.

```
vectorizer = TfidfVectorizer(max_df=0.5,min_df=2,stop_words='english')
```

This vectorizer has a method called fit_transform, which takes a list of strings and then returns a two-dimensional array in which each row represents one article.

```
X = vectorizer.fit_transform(BlogPosts)
X.shape
```

```
(2804, 13220)
```

The shape of the X is (2804, 13220).

Here,

- 2804 represents the number of articles in the corpus.
- 13220 represents the number of distinct words which are present in all articles.

3. Train KMeans clustering algorithm –

Clustering:

Here we are given a large group of articles, and we need to divide these articles into clusters or groups on the basis of some common attributes. We want all articles which represent some particular theme to be put into one group. This is a classic example of a clustering problem.

Whenever you encounter a large number of items and the objective is to divide them into groups based on some measure of similarity, you're basically solving a clustering problem.

The end objective of clustering is to create groups such that items within one group are similar to one another and items which are in different groups are dissimilar to one another.

In other words, if you have some metric called similarity, which measures how similar items are to one another, you want to maximize the intra cluster similarity, maximize the

similarity of items within a cluster. If you want to minimize inter cluster similarity, reduce the similarity between items which are in different clusters.

sklearn.cluster –

Scikit-learn has a built-in module for clustering called the sklearn.cluster module.

Within this module, we have a class for the K-Means clustering algorithm. So we can import that class and instantiate a K-Means clustering object. This sets up the algorithm with all the parameters -

n_clusters is the parameter that is used to specify the number of clusters, which here is three, because we want to divide our articles into three groups.

The init parameter specifies an algorithm to help choose the initial centroids or means in such a way that we can find the relevant clusters with a minimum number of iterations.

We're also specifying the maximum number of iterations, so in case the algorithm does not reach convergence until the 100th iteration, then it will stop at that point.

n_init - Number of times the k-means algorithm will be run with different centroid seeds. Default value is 10.

KMeans –

The k-means clustering method is an unsupervised machine learning technique used to identify clusters of data objects in a dataset. There are many different types of clustering methods, but k-means is one of the oldest and most approachable. These traits make implementing k-means clustering in Python reasonably straightforward, even for novice programmers and data scientists.

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 100, n_init = 1, verbose = True)
```

To perform K-Means clustering, we take our documents represented in TF-IDF, which is the array X, and pass it to the fit method of the K-Means clustering algorithm.

```
km.fit(X)
```

```
Initialization complete
Iteration 0, inertia 5295.914589743915
Iteration 1, inertia 2687.4987712552775
Iteration 2, inertia 2678.3093032168936
Iteration 3, inertia 2674.334183893594
Iteration 4, inertia 2672.5981603897226
Iteration 5, inertia 2671.7974139078005
Iteration 6, inertia 2671.407321803663
Iteration 7, inertia 2671.168635346572
Iteration 8, inertia 2670.9556923588416
Iteration 9, inertia 2670.7850135357794
Iteration 10, inertia 2670.597502500471
Iteration 11, inertia 2670.248381642883
Iteration 12, inertia 2669.509943971192
Iteration 13, inertia 2668.7859783362255
Iteration 14, inertia 2668.6061055744067
Iteration 15, inertia 2668.502079302407
Iteration 16, inertia 2668.337900942934
Iteration 17, inertia 2668.143618790707
Iteration 18, inertia 2667.9754304677153
Iteration 19, inertia 2667.841883194663
Iteration 20, inertia 2667.7369039641226
Iteration 21, inertia 2667.6842434805044
Iteration 22, inertia 2667.6614510483005
Iteration 23, inertia 2667.6520407814787
Iteration 24, inertia 2667.642974913155
Iteration 25, inertia 2667.639592659972
Converged at iteration 25: strict convergence.
KMeans(max_iter=100, n_clusters=3, n_init=1, verbose=True)
```

Every document in our array X has now been assigned a number, which represents the cluster to which it belongs. These numbers are stored in the array labels, which is an attribute of the K-Means object.

We also have the counts which represent how many articles are present in each cluster.

NumPy –

NumPy is a Python library that provides a simple yet powerful data structure: the n-dimensional array. This is the foundation on which almost all the power of Python's data science toolkit is built, and learning NumPy is the first step on any Python data scientist's journey.

For getting the count which represents how many articles are present in each cluster, we used numpy.

```
[18] import numpy as np
     np.unique(km.labels_, return_counts=True)

     (array([0, 1, 2], dtype=int32), array([1592,  778,  434]))
```

4. Find out different themes from the clusters

We have to actually look at the articles which are present in each cluster to identify what meaningful theme is represented by each cluster.

So let's find some of the important keywords which occur in each cluster to see if we can articulate what those underlying themes might be.

We'll set up a dictionary called text in which the keys will be the cluster numbers and the values will be the aggregated text across all the articles which are present in that cluster. We'll go through the array of labels, which have the cluster numbers assigned to each document and then collect the text for each document into the corresponding cluster.

The enumerate function converts the array of labels into a list of tuples where the first element is the index of an article. So using the index, we can get the corresponding article from the list BlogPosts. Then we aggregate this text for every article into the corresponding value in the text dictionary.

enumerate -

The enumerate function converts the array of labels into a list of tuples where the first element is the index of an article. So using the index, we can get the corresponding article from the list BlogPosts. Then we aggregate this text for every article into the corresponding value in the text dictionary.

```
km.labels_
```

```
array([1, 0, 1, ..., 0, 0, 1], dtype=int32)
```

```
text={}
for i,cluster in enumerate(km.labels_):
    oneDocument = BlogPosts[i]
    if cluster not in text.keys():
        text[cluster] = oneDocument
    else:
        text[cluster] += oneDocument
```

```
[21] text

{0: 'Quora tests video answers to steal Q&A from YouTube: Newly-minted unicorn Quora has even bigger ambit
 1: 'SoftBank\'s $100 Billion Tech Fund Rankles VCs as Valuations Soar: In the months since Softbank Group
 2: 'Alibaba Profit Lags Estimates on Tax as Sales Defy Slowdown: Alibaba Group Holding Ltd. s earnings la
```

We can use some NLTK functions to find out the most frequent words within each cluster.

NLTK (Natural Language Toolkit) Functions –

nltk -

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

```
import nltk
```

nltk.tokenize -

Tokenizers divide strings into lists of substrings. For example, tokenizers can be used to find the words and punctuation in a string.

We can also operate at the level of sentences, using the sentence tokenizer.

```
from nltk.tokenize import sent_tokenize, word_tokenize
```

nltk.corpus -

The modules in this package provide functions that can be used to read corpus files in a variety of formats. These functions can be used to read both the corpus files that are distributed in the NLTK corpus package, and corpus files that are part of external corpora.

stopwords - Used to check the list of stopwords.

When we find the most frequent words, we don't want to include stop words, so we'll set up a variable to represent all the stop words that we want to ignore.

Along with the standard list of stop words from English and punctuation, we have some additional words which are common to tech news articles.

```
from nltk.corpus import stopwords
```

```
_stopwords = set(stopwords.words('english') + list(punctuation)+["million","billion","year","millions","billions","y/y","'s","''","``"])
```

nltk.probability -

Classes for representing and processing probabilistic information.

The FreqDist class is used to encode "frequency distributions", which count the number of times that each outcome of an experiment occurs.

The bit of code will take the text from each cluster and find out the top 100 words that occur within that text.

We iterate through each cluster, and for each cluster, we take the corresponding text and tokenize it into words. We filter out all the stop words and keep only relevant words.

Then we use the FreqDist function to compute the frequency distribution of the words.

```
from nltk.probability import FreqDist
```

string -

Module for carrying out common string operations.

punctuation -

String of ASCII characters which are considered punctuation characters in the C locale: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~.

```
from string import punctuation
```

heapq -

This module provides an implementation of the heap queue algorithm, also known as the priority queue algorithm.

nlargest -

Return a list with the n largest elements from the dataset defined by iterable. key, if provided, specifies a function of one argument that is used to extract a comparison key from each element in iterable (for example, key=str.lower). Equivalent to: sorted(iterable, key=key, reverse=True)[:n].

We take the nlargest function and pick the top 100 words from this frequency distribution. Along with the top 100 keywords, we also separately store the complete frequency distribution, which will keep the words along with their counts in the dictionary.

```
from heapq import nlargest
```

```
#to find the top 100 words that occurs frequently in a cluster excluding the stop words
keywords = {}
counts={}
for cluster in range(3):
    word_sent = word_tokenize(text[cluster].lower())
    word_sent=[word for word in word_sent if word not in _stopwords]
    freq = FreqDist(word_sent)
    keywords[cluster] = nlargest(100, freq, key=freq.get)
    counts[cluster]=freq
```

```
keywords[0]
```

```
['company',
 'said',
 'google',
 'facebook',
 'new',
 'amazon',
 'apple',
 'users',
 'like',
 'also',
 'app',
 'people',
 'one',
 'data',
 'mobile',
 'service',
 'companies',
 'percent',
 'twitter',
 'would',
 'online',
 'business',
 'last',
 'ads',
 'could',
 'technology',
 'according',
 'video',
 'first',
 'time',
 'make',
 'customers',
 'use',
 'product',
 'internet',
 'pay',
 'sales',
 'years',
 'even',
 'may',
 'much',
 'way',
 'get',
 'social',
 'microsoft',
 'buy',
 'world',
 'including',
 'market',
 'revenue',
 'big',
 'using',
 'still',
 'content',
 'devices',
 'information',
 'system',
 'tech',
 'made',
 'mr.',
 'u.s.',
 'platform',
 'site',
 'says',
 'announced',
 'web',
 'news',
```

Now that we have the 100 most important keywords from each cluster, let's find the 10 keywords which are unique to each cluster.

As we iterate through each cluster, we find the list of other clusters. So if we look at cluster zero, we collect all the keywords which are present in the other clusters and we remove those keys from the list of keywords in our cluster. From the remaining keywords, we pick the top 10 most frequently occurring keywords.

```
#Let's find out the unique keywords from each cluster
unique_keys={}
for cluster in range(3):
    other_clusters=list(set(range(3))-set([cluster]))
    keys_other_clusters=set(keywords[other_clusters[0]]).union(set(keywords[other_clusters[1]]))
    unique=set(keywords[cluster])-keys_other_clusters
    unique_keys[cluster]=nlargest(10, unique, key=counts[cluster].get)
```

```
unique_keys
```

```
{0: ['ads',
  'video',
  'use',
  'product',
  'products',
  'search',
  'apps',
  'ad',
  'pay',
  'way'],
 1: ['uber',
  'india',
  'round',
  'capital',
  'valuation',
  'chinese',
  'funding',
  'raised',
  'investment',
  'public'],
 2: ['quarter',
  'profit',
  'rose',
  'earnings',
  'analysts',
  'cents',
  'fell',
  'per',
  'net',
  'reported']}
```

One of the clusters seems to be related to social media and advertising-related keywords. So articles that covered those themes would reside in this cluster.

Next cluster seems to have words related to price and profit, which are stock performance-related. So articles about the stock performance might be a part of this cluster.

The other cluster has words like round, capital, funding, and valuation. So this is a cluster that deals with startups and the budget and investments they are getting. So any news articles related to these topics would reside within this cluster.

5. Train K Nearest Neighbors classifier

Now that we have different themes that we have identified from our body of articles, we can take any new article and then assign one of these themes to that article.

Our classification problem statement here is that given an article, we want to pass it to a classifier, and the output should be one of the themes, theme one, two, or three that we have identified from our clustering step.

We take all of our historical data, which is our body of articles, and represent it using the TF-IDF representation. These articles have labels already assigned to them by the clustering step. This training data along with the labels is fed to a standard algorithm in the training step, and that creates the model that can be used in the test step.

sklearn.neighbors -

sklearn.neighbors provides functionality for unsupervised and supervised neighbors-based learning methods. Unsupervised nearest neighbors is the foundation of many other learning methods, notably manifold learning and spectral clustering. Supervised neighbors-based learning comes in two flavors: classification for data with discrete labels, and regression for data with continuous labels.

KNeighborsClassifier -

Classifier implementing the k-nearest neighbors vote.

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X,km.labels_)
```

```
KNeighborsClassifier(n_neighbors=3)
```

We have developed our model to predict the theme of the articles and stored it in a classifier variable.

Results

In this section we can test our model on a different article and check its performance.

Let's take an article related to social media from a random website and assign it to a variable article.

article = "With Snapchat, Twitter, Instagram, and even Facebook, teens are in constant communication. While many will argue that it is important to have this connection outside of face-to-face interaction, studies have said that eliminating in-person communication can impair critical social skills. But what does this really mean? When teens receive messages via social media, all they see is a screen. They can't see the person on the other side. They are oblivious to the other person's reaction. Body language, facial expression, even tone of voice, are removed from the conversation. This is why teens choose to send that risky message or that mean reply, things they wouldn't ever say to someone's face. Of course, this creates a problem. Not only are the simplest parts of communication invisible, but it has become easier and easier to hide behind a mask on social media. Social media allows anyone to be anonymous, identity erased. Cyberbullies can create fake accounts, which allows the bully to say anything without facing consequences. Statistics say that one half of teens have been victims of bullying online and one third have sent rude or harassing comments via social media, (Teens, Social Media & Technology Overview 2015). When teens don't know the person harassing them it is harder to ask for help or tell an adult."

Now before we parse our test article to the KNeighborsClassifier, we need to represent that article using the TF-IDF representation. So we use vectorizer.transform to convert the article into the TF-IDF form.

The test variable will be an array which has one row and as many columns as there are the total number of words in our corpus.

```
test=vectorizer.transform([article.encode('ascii',errors='ignore')])
```

Let's check the shape of the test variable.

```
test.shape
```

```
(1, 13220)
```

Now we can use the predict method of the classifier and that will assign a theme to our article.

```
classifier.predict(test)
```

```
array([1], dtype=int32)
```

Here we can see that the article is assigned to a cluster that describes the social media and advertising cluster.

Our model is perfectly predicting the theme of the articles and is ready to use.

Conclusions and Future Work

The main idea of this project is to save the time of the users by predicting the theme of the articles they are going to read. Our model is trained only with the articles related to the technology and with 2084 articles to check the performance. So, this model is restricted to 3 different themes of articles.

To use this model for different themes, we need to gather a large corpus of articles with different themes and then find out the clusters of those articles by training it with KMeans clustering algorithm. Finally, training a classifier with all the corpus of articles with different themes and then model will predict the right theme for a given article.

References

GeeksforGeeks (2021, October). *Python Urllib Module*. https://www.geeksforgeeks.org/python-urllib-module/

Swetha Kolalapudi (2016, Nov 13). *Getting started with Natural Language Processing with Python.* https://app.pluralsight.com/library/courses/python-natural-language-processing/table-of-contents

GeeksforGeeks (2021, May). *Implementing Web Scraping in Python with BeautifulSoup.* https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/

pypi.org. beautifulsoup4. https://pypi.org/project/beautifulsoup4/

scikit-learn.org. *TfidfVectorizer*. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Kevin Arvai (2020, July). *K-Means Clustering in Python: A Practical Guide*. https://realpython.com/k-means-clustering-python/

scikit-learn.org. *KMeans*. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

scikit-learn.org. *Nearest Neighbors*. https://scikit-learn.org/stable/modules/neighbors.html

Ryan Palo (2021, January). *NumPy Tutorial: Your First Steps Into Data Science in Python*. https://realpython.com/numpy-tutorial/

nltk.org. *nltk.tokenize package*. https://www.nltk.org/api/nltk.tokenize.html

nltk.org. *nltk.corpus package*. https://www.nltk.org/api/nltk.corpus.html

nltk.org. *nltk.probability package*. https://www.nltk.org/_modules/nltk/probability.html

Leodanis Pozo Ramos (2021, July). *Python's collections: A Buffet of Specialized Data Types*. https://realpython.com/python-collections-module/

docs.python.org. *string- Common string operations*. https://docs.python.org/3/library/string.html

docs.python.org. *heapq- Heap queue algorithm*. https://docs.python.org/3/library/heapq.html