# Technical Solution

## Training Model:

I used the standard stack for machine learning (helping functions from sci-kit learn) to train and test my model. I used this stack as I read about its effectiveness and was then happy with the end result:

A screenshot of a cell phone

Description automatically generated

I imported numpy, a library that does mathematical calculations including linear algebra, matrices and vectors multiplication, that is used by the Multinomial Naive Bayes algorithm. It is an optimised array that is used specifically for number calculations, unlike standard python lists.

Pandas is a library that allows you to manipulate with tabular memory data/ lists of records, each column being your numpy array.

The train\_test\_split helps to split your data set into a training set and a data set (a helper function). You provide the ratio, by which the data is split.

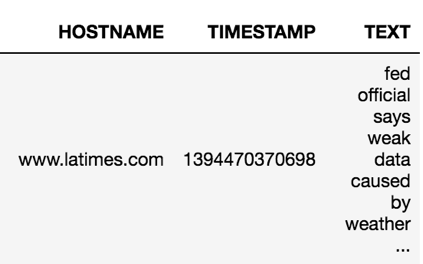
The CountVectoriser function is a common helper method for text processing: turning text into vectors. When given a list of headlines, it finds all unique words across all headlines, which will form a vector space of high dimensionality. After that, the CountVectoriser, turns each headline into a vector in that space, where in each position (word), you have the total of that word repetition in that headline.

The LabelEncoder assigns a number to each unique label to encode each unique category with a number.

## Normalising Text:

Firstly, to use the data in text processing, we need to clean the data. The data was taken from the News API service. Cleaning the data meant stripping all punctuation, replacing multiple spacing with one space and converting to lowercase. This must be done to more accurately vectorise the data.A screenshot of a cell phone

Description automatically generated

**A screenshot of a cell phone

Description automatically generated**Next were showing the pandas dataframe built from csv file:

Where the first column is the original headline and the last in the normalised headline ready to be vectorised.

A screenshot of a social media post

Description automatically generatedNext I vectorised the TEXT column and encoded the category column as a list of whole integers and split the data into a training set and a test set.

When printed, the results outputted is the length of all vectors (number of total unique elements is the length). After our headlines were vectorised, I used the Multinomial Naïve Bayes algorithm to train and test my data, getting a whole 92% accuracy. After this I downloaded my model and vectorizer as a binary file so I then can use my trained model in visual studio.A screenshot of a cell phone

Description automatically generated

## The Classifier:

After having a trained model and extra keywords, I was able to create a full news classifier that was able to predict all 9 categories.

It consists of multiple parts. The first part is the main classifier. Here I will download my model and my vectorizer, ready to work on real data. I will also load my keyterms and create a function that will load the terms. The class will have a separate function for loading terms, predicting categories and matching the keywords (two-grams and three-grams also) from the keyword files and the actual headlines. The class should them return an encoded category. I then created the class and combined it with other elements to complete my news classifying task. I also created a normalising function to get my data prepared for vectorisation.

import pickle

import csv

import re

def normalize\_text(s):

s = s.lower()

s = re.sub(r'\s\W',' ',s)

s = re.sub(r'\W\s',' ',s)

s = re.sub(r'\s+',' ',s)

return s

class Classifier:

def \_\_init\_\_(self, model\_filename='model/nb.model', vectoriser\_filename='model/nb.vectorizer'):

self.model = pickle.load(open(model\_filename, 'rb'))

self.vectoriser = pickle.load(open(vectoriser\_filename, 'rb'))

self.env\_terms = self.load\_terms('model/env\_terms.csv', lambda row: row[1])

self.lgbt\_terms = self.load\_terms('model/lgbt\_terms.csv', lambda row: row[0])

self.youth\_terms = self.load\_terms('model/youth\_terms.csv', lambda row: row[0])

def load\_terms(self, filename, extract):

terms = set()

with open(filename) as f:

reader = csv.reader(f)

for row in reader:

terms.add(extract(row).lower())

return terms

Next, I created the predict function, that normalises the text, vectorises the headlines and then predicts it using the .predict() function that is included with the ML model. It will then return 0,1,2,3 if the headlines do not contain any queries from keyword files. The numbers 1,2,3 and 4 correspond to the 4 categories that the model can categorise. If there are other keywords from the keyword file, it will overwrite the category to the category of the keywords. I decided to overwrite the category instead of having 2 to prevent too many duplicated events/headlines in the wordcloud. It also doesn’t matter which category to put the news in as it is in many categories at the same time. The more specific categories of the keywords give a more specific category of the news. If it has keywords from multiple files, it will assign the most latter category to the headline for simplicity purposes.

def predict(self, headline):

normalised = [normalize\_text(headline)]

x = self.vectoriser.transform(normalised)

y = self.model.predict(x)

if self.match(headline, self.env\_terms):

return 4

if self.match(headline, self.lgbt\_terms):

return 5

if self.match(headline, self.youth\_terms):

return 6

return y[0]

def match(self, headline, terms):

words = normalize\_text(headline).split(' ')

twograms = [words[i] + " " + words[i+1] for i in range(len(words)-1)]

threegrams = [words[i] + " " + words[i+1] + " " + words[i+2] for i in range(len(words)-2)]

words.extend(twograms)

words.extend(threegrams)

for word in words:

if word in terms:

return True

return False

The .match function looks for 1,2 or 3grams from all given keyword files. It will return true or false depending on whether the word is in the file. This will later be used by the predict function that will return 4,5,6 for the last 3 categories. The current categories in this prototype are: environment, youth and lgbt, but this may be changed later on, as I have already decided to not include ‘youth’ as one of my categories.

The second part of classifier is the main file that uses an apikey to load the headlines from newsapi.org and filters them by country and language. In the prototype, it only filers for English and USA. It then applies the prediction of the classifier. It also has to load the classifier from another file and assigns the number returned by the classifier to the category.

from classifier import Classifier

from newsapi import NewsApiClient

categories = ['business', 'entertainment', 'health', 'tech science', 'environment', 'lgbt', 'youth']

def main():

model = Classifier()

#print(model.predict('Apple arcade goes live for iOS 13 beta testers - The Verge'))

newsapi = NewsApiClient(api\_key='16b987ce39464b8296c81b36bc541075')

response = newsapi.get\_top\_headlines(language='en', country='gb')

for article in response['articles']:

headline = article['title']

cat = model.predict(headline)

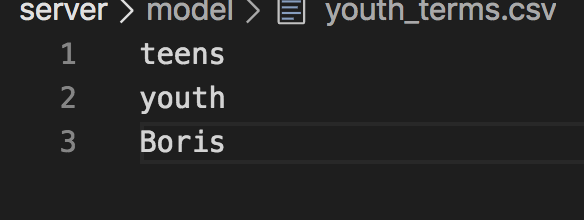
print('{0:<13} | {1}'.format(categories[cat], headline))

if \_\_name\_\_ == '\_\_main\_\_':

main()

These are the keyword files I used for my testing:

A screenshot of a cell phone

Description automatically generatedA screenshot of a social media post with text and a black background

Description automatically generated

Lgbt youth environment

To make sure my application works, I used sample words for lgbt and youth terms.

A screenshot of a cell phone

Description automatically generatedAfter running the program, I got the desired outcome. My headlines were categorized.

The classifier categorised headlines from both the ML model and the keyword files, as both types of categories appeared correctly.

## News Loader:

from newsapi import NewsApiClient

from attrdict import AttrDict

class NewsLoader:

def \_\_init\_\_(self, api\_key):

self.newsapi = NewsApiClient(api\_key=api\_key)

def get\_news(self, county):

response = self.newsapi.get\_top\_headlines(language='en', country='us')

return AttrDict(response)

import sqlite3

from napp.database import create\_database, insert\_news, check\_headline

from napp.news\_loader.news import NewsLoader

import schedule

from datetime import datetime

import time

def main():

conn = sqlite3.connect('database/napp.db')

newsloader = NewsLoader('16b987ce39464b8296c81b36bc541075')

with conn:

create\_database(conn)

country\_code = 'gb'

while True:

load\_news(conn, newsloader, country\_code)

time.sleep(10)

def load\_news(conn, newsloader, country\_code):

print('{} Loading News...'.format(datetime.now()))

response = newsloader.get\_news(country\_code)

for article in response.articles:

headline = article.title

source = article.source.name

url = article.url

if check\_headline(conn, headline) == 0:

news\_id = insert\_news(conn, headline, source, url, country\_code)

print(news\_id)

if \_\_name\_\_ == "\_\_main\_\_":

try:

main()

except KeyboardInterrupt:

print(' Exiting...')

## Database:

import sqlite3

def create\_database(conn):

create\_category\_table = """

CREATE TABLE IF NOT EXISTS Category(

CategoryID INTEGER PRIMARY KEY,

Name VARCHAR NOT NULL

);

"""

create\_event\_table = """

CREATE TABLE IF NOT EXISTS Event(

EventID INTEGER PRIMARY KEY AUTOINCREMENT,

Name VARCHAR NOT NULL,

Summary VARCHAR,

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

"""

create\_news\_table = """

CREATE TABLE IF NOT EXISTS News(

NewsID INTEGER PRIMARY KEY AUTOINCREMENT,

Headline VARCHAR NOT NULL,

Source VARCHAR NOT NULL,

URL VARCHAR NOT NULL,

CountryCode CHAR(3) NOT NULL,

CategoryID INTEGER,

EventID INTEGER,

NewsBody INTEGER,

ScrapedAt TIMESTAMP,

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY(CategoryID) REFERENCES Category(CategoryID),

FOREIGN KEY(EventID) REFERENCES Event(EventID)

);

"""

create\_tweet\_table = """

CREATE TABLE IF NOT EXISTS Tweet(

TweetID INTEGER PRIMARY KEY AUTOINCREMENT,

Tweet VARCHAR NOT NULL,

Hashtag VARCHAR NOT NULL,

URL VARCHAR,

User VARCHAR NOT NULL,

CategoryID INTEGER,

EventID INTEGER,

PublishedAt TIMESTAMP NOT NULL,

CreatedAt TIMESTAMP DEFUALT CURRENT\_TIMESTAMP,

FOREIGN KEY(CategoryID) REFERENCES Category(CategoryID),

FOREIGN KEY(EventID) REFERENCES Event(EventID)

);

"""

conn.execute(create\_category\_table)

conn.execute(create\_event\_table)

conn.execute(create\_news\_table)

conn.execute(create\_tweet\_table)

def insert\_news(conn, headline, source, url, coutry\_code):

sql = """ INSERT INTO News(Headline, Source, URL, CountryCode)

VALUES(?,?,?,?); """

cur = conn.cursor()

cur.execute(sql, (headline, source, url, coutry\_code))

return cur.lastrowid

def check\_headline(conn, headline):

cur = conn.cursor()

cur.execute("SELECT COUNT(\*) FROM News WHERE headline=?", (headline,))

result = cur.fetchone()[0]

return result

## Twitter Loader:

import requests

from requests.auth import AuthBase

from requests.auth import HTTPBasicAuth

# Gets a bearer token

class BearerTokenAuth(AuthBase):

def \_\_init\_\_(self, consumer\_key, consumer\_secret):

self.bearer\_token\_url = "https://api.twitter.com/oauth2/token"

self.consumer\_key = consumer\_key

self.consumer\_secret = consumer\_secret

self.bearer\_token = self.get\_bearer\_token()

def get\_bearer\_token(self):

response = requests.post(

self.bearer\_token\_url,

auth=(self.consumer\_key, self.consumer\_secret),

data={'grant\_type': 'client\_credentials'},

headers={'User-Agent': 'TwitterDevFilteredStreamQuickStartPython'})

if response.status\_code is not 200:

raise Exception(f"Cannot get a Bearer token (HTTP %d): %s" % (response.status\_code, response.text))

body = response.json()

return body['access\_token']

def \_\_call\_\_(self, r):

r.headers['Authorization'] = f"Bearer %s" % self.bearer\_token

r.headers['User-Agent'] = 'TwitterDevFilteredStreamQuickStartPython'

return r

import requests

import json

from twitter\_auth import BearerTokenAuth

import twitter

from napp.database import create\_database

woeids = {

"gb": 23424975,

"us": 23424977

}

trends\_url = "https://api.twitter.com/1.1/trends/place.json"

def get\_trends(auth, country\_code):

woeid = woeids[country\_code]

response = requests.get(trends\_url, auth=auth, params={"id": woeid})

response = response.json()

for trend in response[0]["trends"]:

yield trend["query"]

def get\_tweets(api, keyword):

query = "q={}%20&result\_type=recent&since=2019-11-22&count=100&lang=en"

results = api.GetSearch(raw\_query=query.format(keyword))

return results

def main():

auth = BearerTokenAuth(

consumer\_key="SFkahJJkOcC5rAwcrqQq6cynl",

consumer\_secret="WkVQVWVE09tfIIk6eFon5lfxyGF5llguooZz8JPq9ZQiWgELWj")

api = twitter.Api(consumer\_key="SFkahJJkOcC5rAwcrqQq6cynl",

consumer\_secret="WkVQVWVE09tfIIk6eFon5lfxyGF5llguooZz8JPq9ZQiWgELWj",

access\_token\_key="2752888857-DgQSdhXKDCH4NWynUMmIW9DAcohjflgmvKEQnI4",

access\_token\_secret="sqJ798LQWtgC51HaTxLdcD1RxqgWrxRebN5ikeMqDjQRE")

for trend in get\_trends(auth, "gb"):

tweets = get\_tweets(api, trend)

for tweet in tweets:

print(trend, tweet.text[:80])

if \_\_name\_\_ == "\_\_main\_\_":

main()