Ayush Patel

CS 6120: Natural Language Processing - Prof. Ahmad Uzair

Assignment 2: Text Classification and Neural Network

Total Points: 100 points

In Assignment 2, you will be dealing with text classification using Multinomial Naive Bayes and Neural Networks. You will also be dealing with vector visualization. In the previous assingment you implemented Bag of Words as the feature selection method. However, in this assignment you will be using TF-IDF Vectorization instead of Bag of Words. We recommend starting with this assignment a little early as the datasets are quite large and several parts of the assignment might take long duration to execute.

Question 1 Text Classification

In the first question you will be dealing with AG News Dataset. You are required to implement TF-IDF vectorization from scratch and perform Multinomial Naive Bayes Classification on dataset. You may use appropriate packages or modules for fitting the Multinomial Naive Bayes Model, however, the implementation of the TF-IDF Vectorization should be from the scratch.

In [1]:
```
#importing the libraries

import numpy as np
import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.datasets import fetch_20newsgroups
from pprint import pprint
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import preprocessing
import pandas as pd
import re
import numpy as np
from nltk.tokenize import  word_tokenize
import nltk
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
```

The ag news dataset comprises around 18000 news text in 4 categories.

In [2]:
```
# Read the csv file agnews.csv
mydata_train = pd.read_csv('agnews.csv')
```

In [3]:
```
# Print the news categories in the dataset
print(list(mydata_train.ClassIndex))
```

```
[1, 1, 1, 1, 2, 2, 1, 1, 3, 4, 2, 4, 4, 3, 2, 2, 4, 3, 4, 4, 3, 3, 2, 1,
2, 3, 2, 2, 4, 4, 2, 1, 2, 4, 1, 1, 1, 1, 2, 1, 1, 3, 3, 4, 2, 2, 3, 3, 2,
1, 3, 1, 4, 1, 1, 2, 1, 1, 1, 4, 4, 1, 2, 1, 3, 3, 2, 3, 2, 2, 3, 2, 2, 3,
2, 2, 1, 4, 4, 4, 3, 4, 1, 2, 1, 4, 2, 3, 1, 1, 2, 4, 4, 4, 4, 4, 2, 1, 4,
4, 2, 1, 1, 3, 1, 1, 3, 1, 2, 2, 3, 4, 2, 4, 4, 4, 1, 3, 1, 3, 1, 3, 1, 3,
2, 4, 1, 1, 4, 1, 2, 4, 3, 1, 4, 2, 1, 3, 3, 3, 4, 1, 3, 3, 2, 1, 3, 3, 2,
4, 4, 2, 3, 4, 3, 1, 3, 3, 2, 3, 1, 1, 2, 2, 2, 4, 3, 2, 2, 3, 1, 3, 3, 2,
2, 3, 1, 2, 4, 1, 1, 3, 3, 2, 4, 3, 3, 1, 3, 2, 2, 1, 3, 4, 1, 2, 1, 3, 2,
1, 2, 1, 4, 1, 1, 3, 3, 3, 2, 3, 2, 4, 2, 2, 2, 1, 1, 3, 1, 2, 4, 4, 2, 3,
2, 2, 3, 2, 1, 3, 3, 2, 4, 2, 1, 4, 1, 1, 1, 4, 1, 1, 3, 4, 2, 3, 1, 3, 2,
4, 3, 3, 1, 3, 2, 4, 4, 4, 2, 1, 1, 1, 2, 3, 4, 1, 2, 3, 2, 1, 4, 2, 2, 3,
4, 3, 3, 3, 3, 4, 2, 1, 1, 3, 3, 2, 3, 4, 3, 3, 4, 3, 4, 1, 2, 2, 4, 4, 4,
3, 1, 4, 4, 4, 4, 2, 3, 3, 3, 4, 1, 3, 4, 4, 1, 1, 4, 3, 2, 3, 4, 3, 2, 3,
3, 2, 3, 4, 2, 4, 3, 3, 1, 1, 3, 1, 2, 4, 4, 1, 3, 3, 2, 2, 4, 2, 3, 1, 1,
4, 4, 3, 1, 2, 2, 2, 2, 1, 4, 3, 2, 3, 3, 1, 2, 4, 2, 2, 2, 4, 2, 4, 2, 2,
4, 1, 1, 1, 2, 3, 2, 2, 1, 1, 2, 2, 3, 2, 4, 4, 2, 1, 3, 2, 2, 4, 1, 1, 1,
4, 4, 3, 4, 3, 4, 4, 4, 4, 4, 4, 1, 2, 1, 4, 1, 4, 2, 1, 4, 3, 4, 1, 3, 4,
4, 3, 2, 1, 2, 2, 2, 3, 1, 1, 4, 2, 2, 2, 4, 4, 4, 3, 4, 4, 1, 4, 4, 2, 4,
3, 4, 4, 1, 2, 1, 2, 3, 4, 4, 2, 1, 1, 1, 4, 1, 2, 4, 4, 1, 4, 2, 3, 4, 1,
```

In [4]:
```
# What is the type of 'mydata_train'
print(type(mydata_train))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [5]:
```
# Check the length of the data
print(len(mydata_train.ClassIndex))
print(len(mydata_train.Description))
```

```
18000
18000
```

Expected Output:

18000

18000

Extracting Features from the Dataset (20 Points)

In order to perform machine learning on text documents, we first need to turn the text content into numerical feature vectors.

TF-IDF Vectorization

Our model cannot simply read the text data so we convert it into numerical format. In order to convert the data into numerical format we create vectors from text.

For this particular purpose we could either employ Bag of Words or TF-IDF Vectorization

Bag of Words just creates a set of vectors containing the count of word occurrences in the document (reviews), while the TF-IDF model contains information on the more important words and the less important ones as well.

TF-IDF stands for Term Frequency-Inverse Document Frequency, which instead of giving more weight to words that occur more frequently, it gives a higher weight to words that occur less frequently.

Ref: 6.5 TF-IDF: Weighing terms in the vector section of Dan Jurafsky book
https://web.stanford.edu/~jurafsky/slp3/6.pdf (https://web.stanford.edu/~jurafsky/slp3/6.pdf)

TF-IDF = Term Frequency (TF) * Inverse Document Frequency (IDF)

Term Frequency is the measure of the frequency of words in a document. It is the ratio of the number of times the word appears in a document compared to the total number of words in that document.

The words that occur rarely in the corpus have a high IDF score. It is the log of the ratio of the number of documents to the number of documents containing the word.

idf(t) = log(N/(df + 1))

In [6]:
```
text = mydata_train.Description
```

Preprocessing the Corpus

In [7]:
```
# Preprocessing the data
import string
from tkinter import _flatten

lines = []
word_list = []

for line in text:
    rmv_punc = "".join([w.lower() for w in line if not w in string.punctuation
    token_line = word_tokenize(rmv_punc)
    l = [t for t in token_line if t.isalpha()]
    lines.append(l)

# Make sure the word_list contains unique tokens

word_list = " ".join(list(_flatten(lines)))
word_list = word_tokenize(word_list)
word_list = list(np.unique(np.array(word_list)))

# Calculate the total documents present in the corpus
total_docs = len(text)

# Create a dictionary to keep track of index of each word
dict_idx = {}

for i in range(len(word_list)):
    dict_idx[word_list[i]] = i
```

In [8]:
```
# Create a frequency dictionary

def frequency_dict(lines):
    '''
    lines: list containing all the tokens
    ---
    freq_word: returns a dictionary which keeps the count of the number of doc
    '''

    freq_word = {}

    for w in word_list:
        freq_word[w] = 0

        for l in lines:
            if w in l:
                freq_word[w] += 1

    return freq_word
```

In [9]:
```
# Create a dictionary containing the frequency of words utilizing the 'frequen

# Expect this chunk to take a comparatively longer time to execute since our d

freq_word = frequency_dict(lines)

freq_word
```

Out[9]:

```
{'a': 10541,
 'aac': 1,
 'aain': 1,
 'aal': 1,
 'aapl': 4,
 'aaplo': 1,
 'aaron': 14,
 'ab': 4,
 'ababa': 2,
 'abacha': 1,
 'abandon': 15,
 'abandoned': 20,
 'abandoning': 8,
 'abandonment': 1,
 'abandonmenta': 1,
 'abandons': 1,
 'abankruptcy': 1,
 'abare': 1,
 'abarrel': 2,
 'abated': 1,
 'abbas': 18,
 'abbey': 10,
 'abbott': 2,
 'abbreviated': 1,
 'abby': 2,
 'abc': 3,
 'abcs': 1,
 'abctv': 2,
 'abdicate': 2,
 'abdicated': 2,
 'abdicatethe': 1,
 'abdicating': 1,
 'abdication': 1,
 'abdomen': 1,
 'abdominal': 1,
 'abdoulaye': 1,
 'abducted': 16,
 'abduction': 3,
 'abductions': 1,
 'abductors': 2,
 'abdul': 6,
 'abdullah': 11,
 'aber': 2,
 'abercrombie': 3,
 'aberdeen': 3,
 'abetting': 1,
 'abi': 1,
 'abide': 2,
 'abidine': 1,
 'abidjan': 6,
 'abigger': 1,
 'abilities': 1,
 'ability': 38,
 'abimael': 3,
 'abject': 1,
 'abkhazia': 1,
 'abl': 1,
```

```
 'ablaze': 2,
 'able': 70,
 'ablebodied': 1,
 'abn': 2,
 'aboard': 21,
 'abolish': 1,
 'abolishing': 1,
 'abonns': 1,
 'aboriginal': 1,
 'aborted': 1,
 'abortionrights': 2,
 'abortions': 1,
 'abortive': 3,
 'abound': 1,
 'about': 860,
 'aboutface': 1,
 'aboutpresident': 1,
 'above': 93,
 'abovedjiboutis': 1,
 'abovemedian': 1,
 'abraham': 1,
 'abramovich': 3,
 'abreast': 2,
 'abriton': 1,
 'abroad': 10,
 'abroadaccording': 1,
 'abrupt': 4,
 'abruptly': 9,
 'abs': 1,
 'absa': 2,
 'absence': 15,
 'absent': 2,
 'absentee': 2,
 'absenteeballot': 1,
 'absenteeism': 1,
 'absentia': 1,
 'absolute': 3,
 'absolutely': 1,
 'absorb': 3,
 'absorbed': 2,
 'absorbing': 1,
 'absorbs': 3,
 'abstain': 1,
 'abstained': 1,
 'abstinence': 1,
 'abstracts': 2,
 'absurd': 2,
 'abt': 1,
 'abu': 57,
 'abuja': 19,
 'abukar': 1,
 'abul': 1,
 'abumusab': 2,
 'abundance': 1,
 'abuse': 31,
 'abused': 9,
 'abuses': 13,
```

```
 'abuset': 1,
 'abusing': 4,
 'abusive': 1,
 'abusy': 1,
 'abuzaineh': 1,
 'abuzz': 2,
 'abyss': 1,
 'ac': 17,
 'academic': 14,
 'academics': 1,
 'academies': 2,
 'academy': 5,
 'acadian': 1,
 'acargo': 1,
 'acc': 7,
 'accelerate': 11,
 'accelerated': 4,
 'accelerating': 7,
 'accelerator': 1,
 'accept': 30,
 'acceptability': 1,
 'acceptable': 2,
 'acceptance': 10,
 'accepted': 35,
 'accepting': 8,
 'accepts': 3,
 'access': 105,
 'accessed': 3,
 'accessibility': 1,
 'accessible': 4,
 'accessing': 3,
 'accession': 4,
 'accessories': 3,
 'accessory': 1,
 'accident': 18,
 'accidents': 3,
 'acclaim': 3,
 'acclaimed': 1,
 'acclaimis': 1,
 'accolades': 2,
 'accommodate': 2,
 'accommodates': 1,
 'accommodations': 1,
 'accompanied': 2,
 'accompany': 2,
 'accompanying': 3,
 'accomplish': 2,
 'accomplished': 7,
 'accomplishment': 2,
 'accomplishments': 3,
 'accoona': 1,
 'accor': 2,
 'accord': 7,
 'according': 350,
 'accordingly': 1,
 'accords': 3,
 'account': 34,
```

```
 'accountability': 3,
 'accountable': 1,
 'accountant': 5,
 'accountants': 2,
 'accounted': 5,
 'accounting': 91,
 'accountingfraud': 1,
 'accountingpractices': 1,
 'accountingwebcom': 1,
 'accounts': 36,
 'accra': 1,
 'accredo': 1,
 'accrue': 1,
 'accrued': 1,
 'accumulated': 1,
 'accumulative': 1,
 'accuracy': 3,
 'accurate': 2,
 'accurately': 2,
 'accusation': 1,
 'accusations': 15,
 'accuse': 6,
 'accused': 122,
 'accuser': 3,
 'accuses': 10,
 'accusing': 26,
 'accustomed': 2,
 'acdo': 1,
 'ace': 18,
 'aced': 4,
 'aceh': 4,
 'acer': 2,
 'aceremony': 1,
 'acers': 1,
 'aces': 2,
 'ache': 1,
 'achievable': 1,
 'achieve': 6,
 'achieved': 8,
 'achievement': 7,
 'achievements': 4,
 'achieves': 1,
 'achieving': 2,
 'achievment': 1,
 'achilles': 3,
 'achin': 1,
 'acia': 1,
 'acid': 2,
 'acidic': 1,
 'acis': 1,
 'acknowledged': 13,
 'acknowledgement': 1,
 'acknowledges': 1,
 'acknowledging': 1,
 'aclass': 1,
 'aclosed': 1,
 'acorns': 1,
```

```
 'acquire': 60,
 'acquired': 36,
 'acquires': 1,
 'acquiring': 12,
 'acquisition': 71,
 'acquisitions': 13,
 'acquittal': 3,
 'acquitted': 7,
 'acquitting': 1,
 'acres': 1,
 'acrobat': 6,
 'acrobatically': 1,
 'acropolis': 1,
 'across': 143,
 'acrosshas': 1,
 'acrosstheboard': 3,
 'act': 53,
 'acted': 6,
 'actimmune': 1,
 'acting': 8,
 'action': 115,
 'actionfilled': 1,
 'actions': 19,
 'activate': 1,
 'activated': 11,
 'activation': 1,
 'active': 13,
 'activeduty': 1,
 'activehome': 1,
 'actively': 2,
 'activision': 3,
 'activist': 19,
 'activists': 21,
 'activities': 38,
 'activitiesbut': 1,
 'activity': 49,
 'acto': 1,
 'actonboxboro': 1,
 'actor': 7,
 'actors': 4,
 'actress': 3,
 'acts': 10,
 'actual': 5,
 'actually': 19,
 'actuarial': 2,
 'actuate': 1,
 'actuator': 1,
 'acura': 1,
 'acute': 4,
 'ad': 24,
 'ada': 1,
 'adage': 3,
 'adam': 23,
 'adamant': 1,
 'adams': 7,
 'adamsoshiomhole': 1,
 'adamss': 1,
```

```
 'adapt': 9,
 'adaptec': 1,
 'adapted': 1,
 'adapter': 1,
 'adapters': 7,
 'adapting': 1,
 'adaptive': 1,
 'adaptor': 1,
 'adaptors': 1,
 'adb': 3,
 'add': 38,
 'added': 80,
 'adderall': 1,
 'addict': 2,
 'addicted': 2,
 'addiction': 4,
 'addictive': 2,
 'addicts': 1,
 'adding': 41,
 'addingthat': 1,
 'addis': 2,
 'addition': 22,
 'additional': 37,
 'additionalinternational': 1,
 'additionally': 1,
 'additions': 4,
 'addled': 1,
 'addon': 4,
 'address': 32,
 'addressed': 6,
 'addresses': 9,
 'addressing': 6,
 'adds': 12,
 'adeal': 1,
 'adelaide': 6,
 'adelaideborn': 1,
 'adelphia': 4,
 'adelphiacommunications': 1,
 'aden': 1,
 'adequate': 2,
 'adequately': 3,
 'adewale': 2,
 'adhd': 1,
 'adhere': 1,
 'adheres': 2,
 'adhoc': 1,
 'adi': 1,
 'adidas': 1,
 'adieu': 2,
 'adirondacks': 1,
 'adisai': 1,
 'adjacent': 1,
 'adjourned': 3,
 'adjourning': 1,
 'adjournment': 1,
 'adjourns': 1,
 'adjust': 4,
```

```
 'adjusted': 5,
 'adjusting': 4,
 'adjustment': 1,
 'adjustments': 3,
 'adjusts': 1,
 'administering': 1,
 'administration': 120,
 'administrations': 6,
 'administrative': 9,
 'administrativepersonnel': 1,
 'administrator': 13,
 'administrators': 9,
 'admirable': 1,
 'admiration': 1,
 'admire': 1,
 'admirer': 1,
 'admires': 1,
 'admiring': 1,
 'admission': 5,
 'admissions': 2,
 'admit': 4,
 'admits': 8,
 'admitted': 42,
 'admitting': 9,
 'admonished': 2,
 'adnan': 1,
 'ado': 2,
 'adobe': 17,
 'adobes': 1,
 'adolescence': 2,
 'adolescent': 1,
 'adolescents': 1,
 'adolph': 1,
 'adopt': 11,
 'adopted': 13,
 'adopting': 3,
 'adoption': 6,
 'adorable': 1,
 'adoring': 1,
 'adpowers': 1,
 'adr': 1,
 'adriaanse': 1,
 'adrian': 31,
 'adriano': 4,
 'adriatic': 2,
 'adrift': 3,
 'adroit': 1,
 'adrought': 1,
 'ads': 26,
 'adsense': 2,
 'adt': 4,
 'adtech': 1,
 'adu': 1,
 'adulation': 1,
 'adult': 7,
 'adultery': 2,
 'adulthood': 1,
```

```
 'adultonly': 1,
 'adults': 17,
 'advance': 26,
 'advanced': 73,
 'advancements': 1,
 'advancers': 1,
 'advances': 10,
 'advancing': 6,
 'advanta': 2,
 'advantage': 31,
 'advantagehe': 1,
 'advantest': 2,
 'advent': 1,
 'adventure': 6,
 'adventurers': 2,
 'adversely': 1,
 'adversity': 2,
 'advert': 2,
 'advertise': 1,
 'advertised': 2,
 'advertisement': 3,
 'advertisements': 7,
 'advertiser': 1,
 'advertisers': 4,
 'advertisersponsoredsearch': 1,
 'advertising': 60,
 'advertisingblogger': 1,
 'advertizing': 1,
 'advice': 12,
 'advise': 5,
 'advised': 5,
 'adviser': 9,
 'advisers': 11,
 'advising': 4,
 'advisor': 2,
 'advisories': 3,
 'advisors': 4,
 'advisory': 9,
 'advo': 1,
 'advocacy': 4,
 'advocate': 2,
 'advocates': 8,
 'advocating': 1,
 'adware': 5,
 'adweekcom': 1,
 'adwords': 2,
 'adwordshave': 1,
 'aere': 4,
 'aerial': 6,
 'aeromxico': 1,
 'aeronautic': 1,
 'aeronautical': 1,
 'aeronautics': 4,
 'aeroproducer': 1,
 'aerospace': 14,
 'aether': 1,
 'aeuronautics': 1,
```

```
 'afa': 1,
 'afamiliar': 1,
 'afar': 5,
 'afc': 6,
 'afederal': 1,
 'affair': 7,
 'affairs': 19,
 'affect': 14,
 'affected': 7,
 'affecting': 8,
 'affection': 1,
 'affections': 1,
 'affects': 5,
 'affidavit': 1,
 'affiliate': 7,
 'affiliated': 2,
 'affiliateebay': 1,
 'affiliates': 6,
 'affirming': 1,
 'affix': 1,
 'affixed': 1,
 'affleck': 2,
 'afflicting': 3,
 'affluent': 3,
 'afford': 8,
 'affordable': 11,
 'afghan': 58,
 'afghanistan': 96,
 'afghanistans': 19,
 'afghans': 8,
 'afl': 1,
 'aflcio': 2,
 'afloat': 1,
 'afp': 378,
 'afpone': 1,
 'afraid': 3,
 'africa': 82,
 'africacalled': 1,
 'african': 89,
 'africanized': 1,
 'africanregistered': 1,
 'africans': 5,
 'africanunion': 1,
 'africas': 12,
 'afrol': 2,
 'aft': 1,
 'after': 1836,
 'afterglow': 1,
 'afterhours': 3,
 'aftermath': 9,
 'afternoon': 55,
 'afternoons': 3,
 'afterschool': 3,
 'aftershocks': 5,
 'afterspinning': 1,
 'afterstorming': 1,
 'afterstorms': 1,
```

```
 'afterthe': 2,
 'afterthey': 1,
 'afterthought': 1,
 'aftertwo': 1,
 'afterus': 1,
 'afterward': 3,
 'afterwards': 1,
 'afterweatherrelated': 1,
 'afwerki': 1,
 'afx': 5,
 'afxeastman': 1,
 'ag': 33,
 'again': 164,
 'againasked': 1,
 'against': 851,
 'againstmajor': 1,
 'againstseemingly': 1,
 'againstthe': 2,
 'agali': 1,
 'agassi': 17,
 'agassis': 1,
 'agate': 1,
 'age': 38,
 'aged': 6,
 'agediscrimination': 1,
 'ageing': 1,
 'agence': 4,
 'agencies': 38,
 'agency': 148,
 'agencys': 7,
 'agencywill': 1,
 'agenda': 20,
 'agendas': 1,
 'ageneral': 1,
 'agent': 35,
 'agents': 20,
 'agere': 1,
 'ages': 1,
 'agfagevaert': 1,
 'agganis': 1,
 'aggies': 1,
 'aggravated': 2,
 'aggravating': 1,
 'aggregate': 1,
 'aggressive': 27,
 'aggressively': 10,
 'aggressor': 1,
 'aghahowa': 1,
 'agi': 2,
 'agila': 1,
 'agility': 1,
 'agilysys': 1,
 'aging': 16,
 'agingand': 1,
 'agitating': 1,
 'agitation': 1,
 'agm': 1,
```

```
 'agni': 1,
 'ago': 246,
 'agonize': 1,
 'agonized': 1,
 'agonizing': 1,
 'agoos': 1,
 'agp': 1,
 'agranew': 1,
 'agree': 36,
 'agreed': 272,
 'agreedin': 1,
 'agreedlate': 1,
 'agreedto': 2,
 'agreeing': 12,
 'agreement': 173,
 'agreements': 14,
 'agrees': 8,
 'agricultural': 1,
 'agriculture': 11,
 'ags': 1,
 'agustin': 1,
 'ah': 1,
 'ahead': 193,
 'aheadltagtltbrgtsolaris': 2,
 'aheadof': 1,
 'aheads': 1,
 'ahern': 4,
 'ahhhh': 1,
 'ahigher': 2,
 'ahk': 1,
 'ahlah': 1,
 'ahmad': 5,
 'ahman': 3,
 'ahmed': 12,
 'ahold': 6,
 'ahotshot': 1,
 'ahref': 1,
 'ahronot': 1,
 'ahsanul': 1,
 'ahydroelectric': 1,
 'ai': 3,
 'aibo': 1,
 'aid': 86,
 'aide': 14,
 'aided': 3,
 'aider': 1,
 'aides': 12,
 'aiding': 4,
 'aids': 10,
 'aig': 2,
 'aikman': 1,
 'aiko': 1,
 'ailing': 18,
 'aillet': 1,
 'ailment': 3,
 'aim': 17,
 'aimbot': 1,
```

```
 'aimed': 99,
 'aiming': 15,
 'aims': 33,
 'ain': 2,
 'ainge': 3,
 'ainsworth': 1,
 'aint': 2,
 'aipac': 1,
 'air': 198,
 'airasia': 2,
 'airbase': 1,
 'airborne': 1,
 'airbus': 37,
 'airbuss': 1,
 'aircover': 1,
 'aircraft': 55,
 'aircraftdemand': 1,
 'aircraftgovernment': 1,
 'aircraftmaintenance': 1,
 'aircraftmaker': 1,
 'aircraftmakers': 1,
 'aircrafts': 1,
 'airdrop': 1,
 'aired': 11,
 'aires': 9,
 'airespace': 1,
 'airexpress': 3,
 'airfares': 2,
 'airflow': 1,
 'airforce': 1,
 'airing': 4,
 'airlift': 2,
 'airlifted': 3,
 'airlifts': 1,
 'airline': 98,
 'airlineattacks': 1,
 'airliner': 4,
 'airliners': 5,
 'airlines': 105,
 'airlinescorp': 1,
 'airplane': 2,
 'airplanes': 4,
 'airpolluting': 1,
 'airport': 58,
 'airports': 9,
 'airpower': 1,
 'airshipping': 1,
 'airstrike': 3,
 'airstrikes': 4,
 'airtel': 1,
 'airtran': 4,
 'airwave': 1,
 'airwaves': 9,
 'airways': 73,
 'aishwariya': 1,
 'aisle': 1,
 'aisles': 1,
```

```
 'aix': 1,
 'aiyar': 1,
 'aiyegbeni': 1,
 'aj': 4,
 'ajax': 10,
 'ajudge': 1,
 'aka': 3,
 'akerson': 1,
 'akhmad': 1,
 'akhtar': 2,
 'akhtars': 1,
 'akhurst': 1,
 'akihito': 2,
 'akimbo': 1,
 'akland': 1,
 'akram': 1,
 'akron': 8,
 'al': 69,
 'ala': 10,
 'alabama': 11,
 'alabamas': 1,
 'alain': 2,
 'alameda': 3,
 'alamos': 2,
 'alan': 45,
 'alandmark': 1,
 'alaqsa': 2,
 'alarabiya': 4,
 'alarge': 2,
 'alarm': 4,
 'alarmed': 1,
 'alarming': 3,
 'alarmingly': 1,
 'alarms': 2,
 'alaska': 12,
 'alaskas': 1,
 'alastair': 2,
 'alaxala': 1,
 'alazzawi': 1,
 'albacete': 3,
 'albalah': 1,
 'albania': 1,
 'albanian': 4,
 'albanians': 4,
 'albany': 3,
 'albar': 1,
 'albashir': 1,
 'albatrawi': 1,
 'albatrosses': 1,
 'albeit': 5,
 'albert': 12,
 'alberta': 9,
 'albertas': 1,
 'alberto': 3,
 'albertofujimoris': 2,
 'alberts': 1,
 'albertsons': 4,
```

```
'albeshir': 1,
'albion': 5,
'album': 4,
'albums': 3,
'albuquerque': 1,
'alcan': 1,
'alcantara': 1,
'alcatel': 1,
'alcntara': 1,
'alcoa': 7,
'alcohol': 5,
'alcoholic': 1,
'alcs': 1,
'alderson': 2,
'aldosari': 1,
'aldouri': 3,
'alejandro': 1,
'aleksander': 1,
'aleksandr': 1,
'alemany': 1,
'alert': 15,
'alerted': 1,
'alerts': 1,
'alessandro': 6,
'alevel': 1,
'alex': 34,
'alexander': 18,
'alexandra': 1,
'alexandre': 2,
'alexandria': 8,
'alexandros': 1,
'alexei': 2,
'alexy': 1,
'alfallujah': 1,
'alfie': 1,
'alfonseca': 1,
'alfonso': 3,
'alfred': 2,
'alfredo': 2,
'algae': 5,
'algeria': 1,
'algerian': 2,
'algiers': 2,
'algorithm': 1,
'alh': 1,
'alhabib': 1,
'alhaidari': 1,
'ali': 31,
'alice': 1,
'alicia': 7,
'alien': 4,
'aliens': 1,
'alienware': 1,
'alight': 2,
'align': 1,
'aligned': 2,
'aligning': 1,
```

```
 'alike': 2,
 'alina': 2,
 'alioto': 1,
 'alislamiya': 1,
 'alison': 1,
 'alist': 1,
 'alistair': 1,
 'alitalia': 15,
 'alive': 30,
 'aljafari': 1,
 'aljazeera': 17,
 'aljihad': 2,
 'alkaradma': 1,
 'alkhazraji': 1,
 'alkidwa': 1,
 'alkmaar': 2,
 'all': 484,
 'allactivities': 1,
 'allamerican': 2,
 'allan': 4,
 'allardyce': 2,
 'allaround': 12,
 'allawi': 39,
 'allawis': 1,
 'allay': 3,
 'allayed': 1,
 'allback': 1,
 'allbig': 1,
 'allbutunnoticed': 1,
 'allcash': 2,
 'allclear': 1,
 'allconquering': 2,
 'allconsuming': 1,
 'alldigital': 1,
 'allegation': 1,
 'allegations': 38,
 'alleged': 79,
 'allegedgly': 1,
 'allegedly': 39,
 'alleges': 4,
 'allegiance': 1,
 'alleging': 9,
 'allen': 27,
 'allens': 1,
 'allergyfree': 1,
 'alles': 1,
 'allesandro': 1,
 'alleviate': 2,
 'alleviating': 1,
 'alley': 2,
 'alleyne': 2,
 'allgemeine': 1,
 'allgirl': 1,
 'alliance': 40,
 'alliances': 7,
 'allied': 2,
 'allies': 16,
```

```
 'allimportant': 1,
 'allinone': 5,
 'allison': 3,
 'allister': 1,
 'allmale': 1,
 'allmet': 1,
 'allnew': 2,
 'allocated': 3,
 'allocates': 1,
 'allocation': 1,
 'alloracle': 1,
 'allotment': 1,
 'allotted': 1,
 'allout': 5,
 'allow': 91,
 'allowance': 1,
 'allowances': 1,
 'allowed': 37,
 'allowing': 30,
 'allows': 36,
 'allowsowners': 1,
 'allparty': 1,
 'allpro': 7,
 'allpurpose': 1,
 'allround': 4,
 'allrounder': 2,
 'allrussian': 4,
 'allscholastic': 1,
 'allscrip': 1,
 'allshare': 1,
 'allstar': 19,
 'allstars': 1,
 'allstate': 3,
 'allstock': 4,
 'allston': 1,
 'alltel': 1,
 'allterrain': 1,
 'alltheway': 1,
 'alltime': 22,
 'allure': 3,
 'alluring': 1,
 'ally': 15,
 'allyson': 4,
 'alm': 1,
 'alma': 1,
 'almajid': 1,
 'almansour': 2,
 'almasri': 4,
 'almaty': 1,
 'almighty': 1,
 'almond': 1,
 'almonds': 1,
 'almost': 112,
 'almunia': 3,
 'almustapha': 1,
 'alnajaf': 1,
 'alnuaimi': 1,
```

```
'alobeidi': 1,
'aloftier': 1,
'aloisi': 1,
'alone': 20,
'along': 71,
'alongside': 10,
'alonso': 3,
'alonzo': 3,
'alou': 2,
'alp': 1,
'alpha': 2,
'alphabet': 1,
'alphabets': 1,
'alpharma': 1,
'alphonse': 1,
'alpi': 1,
'alpine': 8,
'alps': 3,
'alqa': 1,
'alqaeda': 15,
'alqaedalinked': 4,
'alqaida': 19,
'alqaidalinked': 4,
'alramadi': 1,
'already': 107,
'alreadylowered': 1,
'alsadr': 25,
'alsadrs': 1,
'alshara': 1,
'alsheikh': 1,
'alshuja': 1,
'alsistani': 6,
'also': 177,
'alsorans': 1,
'alstom': 1,
'alstott': 1,
'alta': 3,
'altana': 1,
'alter': 7,
'altered': 5,
'altering': 2,
'alternate': 1,
'alternative': 23,
'alternatives': 5,
'alters': 1,
'althea': 1,
'although': 60,
'altitude': 4,
'alto': 4,
'altogether': 2,
'altoids': 3,
'altrac': 1,
'altria': 4,
'altruistic': 1,
'alumina': 3,
'aluminum': 9,
'alumni': 4,
```

```
 'alvarez': 2,
 'alvaro': 1,
 'alvidrez': 1,
 'alvin': 4,
 'always': 36,
 'alyawar': 3,
 'alyssa': 1,
 'alzarqawi': 18,
 'alzarqawis': 2,
 'alzawahri': 1,
 'alzheimer': 2,
 'alzheimers': 5,
 'am': 39,
 'ama': 1,
 'amajor': 1,
 'amana': 1,
 'amanda': 1,
 'amani': 1,
 'amar': 1,
 'amare': 2,
 'amassed': 2,
 'amateur': 9,
 'amazing': 4,
 'amazon': 10,
 'amazoncom': 17,
 'amazoncouk': 2,
 'amazonian': 1,
 'ambani': 7,
 'ambanis': 2,
 'ambassador': 20,
 'ambassadors': 2,
 ...}
```

In [10]:

```
# Create a function to calculate the Term Frequency

def term_frequency(document, word):
    '''
    document: list containing the entire corpus
    word: word whose term frequency is to be calculated
    ---
    tf: returns term frequency value
    '''

    n = len(document)
    occ = 0

    for w in document:
        if w == word:
            occ += 1

    tf = occ/n

    return tf
```

In [11]:
```
# Create a function to calculate the Inverse Document Frequency

def inverse_df(word):
    '''
    word: word whose inverse document frequency is to be calculated
    ---
    idf: return inverse document frequency value
    '''

    idf = np.log(total_docs/freq_word[word]+1)

    return idf
```

In [12]:
```
#Create a function to combine the term frequencies (TF) and inverse document (

def tfidf(sentence, dict_idx):
    '''
    sentence: list containing the entire corpus
    dict: dictionary keeping track of index of each word
    ---
    tf_idf_vec: returns computed tf-idf
    '''

    n= len(dict_idx)
    tf_idf_vec = np.zeros(n)

    for w in sentence:
        tf = term_frequency(sentence, w)
        idf = inverse_df(w)

        tf_idf_vec[dict_idx[w]] = tf*idf

    return tf_idf_vec
```

In [13]:
```
#Compute the vectors utilizing the 'tfidf' function created above to obtain a

vectors = []

for line in lines:
    v = tfidf(line, dict_idx)
    vectors.append(v)
```

Multinomial Naive Bayes (10 Points)

In [14]:
```
from sklearn.preprocessing import normalize

#Fit a Multinomial Naive Bayes Model on our dataset

data = normalize(vectors, axis=0, norm='max')
clf = MultinomialNB(alpha = 0.01)
clf.fit(data, mydata_train.ClassIndex)
```

Out[14]:
```
MultinomialNB(alpha=0.01)
```

In [15]:
```
#Perform testing on the train dataset

pred = clf.predict(vectors)
```

In [16]:
```
#Calculate the F1 Score and the Accuracy

F1_score = metrics.f1_score(mydata_train.ClassIndex, pred, average = 'macro')
Accuracy = metrics.accuracy_score(mydata_train.ClassIndex, pred)

print("F1 Score: ", F1_score)
print("Accuracy: ", Accuracy)
```

```
F1 Score:  0.9713644404448953
Accuracy:  0.9713888888888889
```

Expected Output:

F1 Score: 0.9604092771164052

Accuracy: 0.9604444444444444

Your accuracy does not have to be exactly the same. This is just to give you an estimate of what could you expect your accuracy to be around.

Question 2 Vector Visualization

In this unsupervised learning task we are going to cluster wikipedia articles into groups using T-SNE visualization after vectorization.

Collect articles from Wikipedia (10 points)

In this section we will download articles from wikipedia and then vectorize them in the next step. You can select somewhat related topics or fetch the articles randomly. (Use dir() and help() functions or refer wikipedia documentation) You may also pick any other data source of your choice instead of wikipedia.

In [17]:
```
# install libraries
!pip install wikipedia
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: wikipedia in c:\users\ayush\appdata\roaming\py
thon\python39\site-packages (1.4.0)
Requirement already satisfied: requests<3.0.0,>=2.0.0 in c:\programdata\anaco
nda3\lib\site-packages (from wikipedia) (2.28.1)
Requirement already satisfied: beautifulsoup4 in c:\programdata\anaconda3\lib
\site-packages (from wikipedia) (4.11.1)
Requirement already satisfied: idna<4,>=2.5 in c:\programdata\anaconda3\lib\s
ite-packages (from requests<3.0.0,>=2.0.0->wikipedia) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anaconda3
\lib\site-packages (from requests<3.0.0,>=2.0.0->wikipedia) (2022.9.14)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\programdata\anacon
da3\lib\site-packages (from requests<3.0.0,>=2.0.0->wikipedia) (1.26.11)
Requirement already satisfied: charset-normalizer<3,>=2 in c:\programdata\ana
conda3\lib\site-packages (from requests<3.0.0,>=2.0.0->wikipedia) (2.0.4)
Requirement already satisfied: soupsieve>1.2 in c:\programdata\anaconda3\lib
\site-packages (from beautifulsoup4->wikipedia) (2.3.1)
```

In [18]:
```
import wikipedia
from wikipedia.exceptions import WikipediaException
from itertools import repeat
'''
 Generate a list of wikipedia article to cluster
 You can maintain a static list of titles or generate them randomly using wiki
 Some topics include:
 ["Northeastern Unversity", "Natural language processing", "Machine learning",
 "Bank of America", "Visa Inc.", "European Central Bank", "Bank", "Financial t
 "Basketball", "Swimming", "Tennis", "Football", "College Football", "Associat

 You can add more topics from different categories so that we have a diverse d
 Ex- About 3+ categories(groups), 3+ topics in each category, 3+ articles in e
'''

# selected topics

group_1 = ["Natural language processing", "Machine learning", "Quantum machine
group_2 = ["Basketball", "Swimming", "Tennis", "Football", "College Football"]
group_3 = ["Bank of America", "Visa Inc.", "European Central Bank", "Bank"]

topics = group_1 + group_2 + group_3

# list of articles to be downloaded
articles = {}
for t in topics:
    A = wikipedia.search(t, results=3)

    if t in group_1:
        i = 1
    elif t in group_2:
        i = 2
    elif t in group_3:
        i = 3

    for a in A:
        articles[a] = i

# download and store articles (summaries) in this variable
data = []
label = []

for a in articles:
    try:
        data.append(wikipedia.summary(a))
        label.append(articles[a])

    except WikipediaException:
        continue
```

Cleaning the Data (5 points)

In this step you will decide whether to clean the data or not. If you choose to clean, you may utilize the clean function from assignment 1.

Question: Why are you (not) choosing to clean the data? Think in terms of whether cleaning data will help in the clustering or not.

Answer(1-3 sentences):

I chose not to clean data because after actually comparing the plotting results with and without clean, I found that unclean data would help more with clustering.

My cleaning steps included removing punctuation and stop words, numbers, etc. I thought this should be more helpful for clustering, but in reality it is not. But the difference between them is not that big.

In [19]:
```
# You can use Assignment 1's clean message function
import re
import nltk
import string
#nltk.download('punkt')
#nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

def clean_message(message):

    stopw = stopwords.words('english')
    Lemma = WordNetLemmatizer()

    message = message.lower()
    message = re.sub(r"http\S+", " ", message)
    message = re.sub(r"www.\S+", " ", message)
    message = re.sub(r"<br />", " ", message)

    rmv_punc = "".join([r for r in message if not r in string.punctuation])
    tokens = word_tokenize(rmv_punc)
    words = [t for t in tokens if not t in stopw and t.isalpha()]

    # Lemmatization
    message_cleaned = " ".join([Lemma.lemmatize(w) for w in words])

    return message
```

Vectorize the articles (5 points)

In this step, we will vectorize the text data. You can use TfidfVectorizer() or countVectorizer() from sklearn library.

In [20]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# # Clean data
# data_cleaned = [clean_message(m) for m in data]

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(data)
```

In [21]:
```
print(X.shape)
```

```
(33, 2018)
```

Sample Output:

(36, 1552)

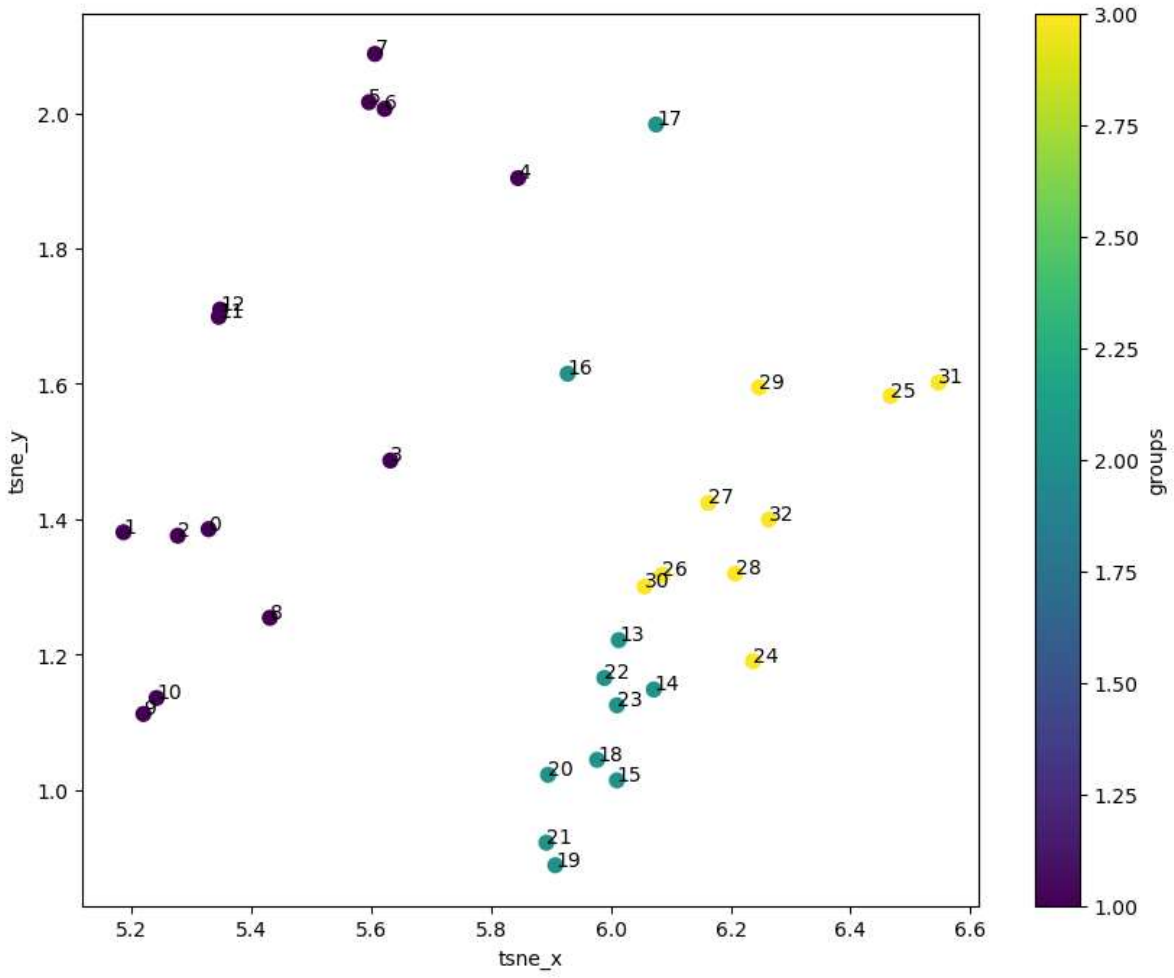
Plot Articles (10 points)

Now we will try to verify the groups of articles using T-SNE from sklearn library.

In [22]:
```
 from sklearn.manifold import TSNE

# call TSNE() to fit the data
X_tsne = TSNE(n_components = 2, init="random").fit_transform(X)
```

Plot and annotate the points with different markers for different expected groups.

In [23]:
```
import matplotlib.pyplot as plt

# get a figure handle
fig,ax = plt.subplots(figsize=(10,8))

X_tsne_df = pd.DataFrame({'tsne_x': X_tsne[:,0], 'tsne_y': X_tsne[:,1], 'group
X_tsne_df.plot.scatter(x = 'tsne_x', y = 'tsne_y', c = 'groups', ax = ax, s =

for i in range(len(X_tsne_df)):
    ax.annotate(i,(X_tsne[:,0][i],X_tsne[:,1][i]))
```



Question: Comment about the categorizion done by T-SNE. Do the articles of related topics cluster together? (5 points)

Answer(1-3 sentences):

After several trials, it's clear from the results presented on the plotting that in most cases, articles on the related topics are relatively clustered together.

Question 3. Building Multinomial Naive Bayes and Neural Networks on Countvectors

We are gonna use Disaster Tweets Datasset for this task. We need to Predict which Tweets are about real disasters and which ones are not.

We are providing data.csv file along with this notebook.

Library Imports and Utility functions

In [24]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
import string
import pandas as pd
import re
#string.punctuation
import nltk
# nltk.download('stopwords')
# nltk.download('wordnet')
# nltk.download('words')

stopword = nltk.corpus.stopwords.words('english')
wn = nltk.WordNetLemmatizer()
ps = nltk.PorterStemmer()
words = set(nltk.corpus.words.words())


def clean_text(text):
  # From the last assignment
    text = text.lower()
    text = re.sub(r"http\S+", "", text)
    text = re.sub(r"www.\S+", "", text)
    text_links_removed = "".join([char for char in text if char not in string.
    text_cleaned = " ".join([word for word in re.split('\W+', text_links_remov
        if word not in stopword])
    text = " ".join([wn.lemmatize(word) for word in re.split('\W+', text_clear
    return text
```

Q) Importing the datasets and do the necessary cleaning and convert the text into the vectors which are mentioned in the below code blocks. (10 points)

In [25]:
```
# Import the data.csv only use 'text' and 'target' columns

data = pd.read_csv("data-1.csv")

# and printout the train.shape and validation.shape

data = data.drop(['keyword','location','id'], axis = 1)

# expected shape of dataset is (7613, 2)

data.shape
```

Out[25]: (7613, 2)

In [26]:
```
data
```

Out[26]:

	text	target
0	Our Deeds are the Reason of this #earthquake M...	1
1	Forest fire near La Ronge Sask. Canada	1
2	All residents asked to 'shelter in place' are ...	1
3	13,000 people receive #wildfires evacuation or...	1
4	Just got sent this photo from Ruby #Alaska as ...	1
...	...	...
7608	Two giant cranes holding a bridge collapse int...	1
7609	@aria_ahrary @TheTawniest The out of control w...	1
7610	M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...	1
7611	Police investigating after an e-bike collided ...	1
7612	The Latest: More Homes Razed by Northern Calif...	1

7613 rows × 2 columns

In [27]:
```
# clean the text in the dataframe using the clean_text function provided above

data["text"] = data["text"].apply(clean_text)
```

In [28]:
```
# initialise count vectorizer from sklearn module with default parameter
Count_vec = CountVectorizer()

# fit on train dataset and transform both train and validation dataset
Count_vec.fit(data)

data_1 = Count_vec.fit_transform(data["text"]).toarray()
```

In [29]:
```
data_1.shape
```

Out[29]:
```
(7613, 16362)
```

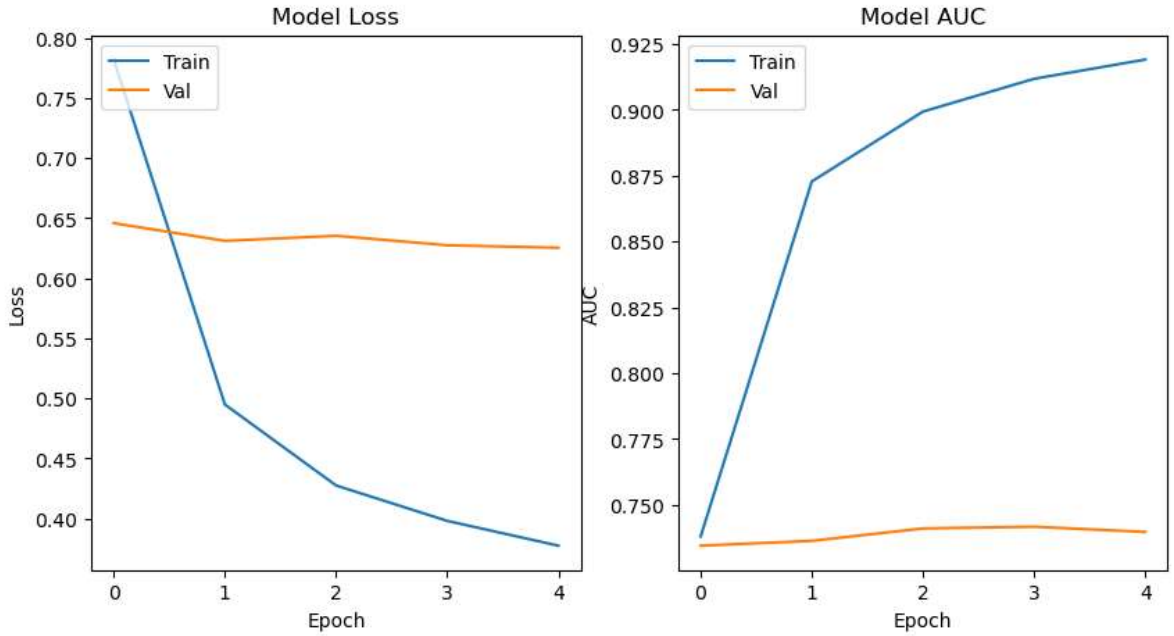
In [30]:
```
# get the values of target column

y = data["target"]
```

Q) Build the neural networks using tensorflow keras by following the below instructions. Evaluate the model on different metrics and comment your observations. (15 points)

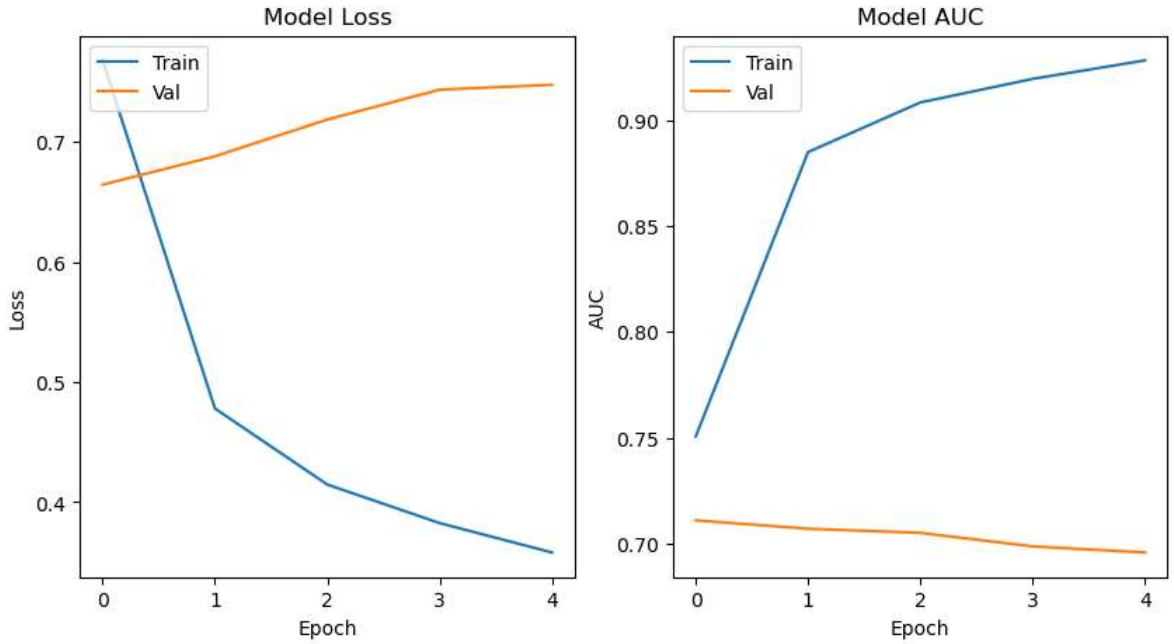
In [31]:
```
import tensorflow as tf
from tensorflow.keras.metrics import AUC

tf.random.set_seed(42)


# complete this linear model in tensorflow
def build_model(X):

    model = tf.keras.models.Sequential()
    # layer 1 : input layer
    input_layer = tf.keras.layers.Input((X.shape[1],))
    model.add(input_layer)

    # layer 2 : add the dense layer with 64 units and relu activation
    model.add(tf.keras.layers.Dense(units = 64, activation = "relu"))

    # layer 3 : add the dropout layer with dropout rate of 0.5
    model.add(tf.keras.layers.Dropout(0.5))

    # layer 4 : add the dense layer with 32 units with tanh activation and wit
    model.add(tf.keras.layers.Dense(units = 32, activation = "tanh", kernel_re

    # layer 5 : add the dropout layer with dropout rate of 0.5
    model.add(tf.keras.layers.Dropout(0.5))

    # layer 6 : add the dense layer with 16 units with tanh activation and wit
    model.add(tf.keras.layers.Dense(units = 16, activation = "tanh", kernel_re

    # layer 7 : add the dropout layer with dropout rate of 0.5
    model.add(tf.keras.layers.Dropout(0.5))

    # layer 8 : output layer with units equal to 1 and activation as sigmoid
    model.add(tf.keras.layers.Dense(units = 1, activation = "sigmoid"))

    # use loss as binary crossentropy, optimizer as rmsprop and evaluate model
    model.compile(optimizer = 'rmsprop', loss = tf.keras.losses.BinaryCrossent

    return model
```

In [32]:
```
# Now we will initialise the stratified K-Fold from sklearn with nsplits as 5
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_curve

skf = StratifiedKFold(n_splits = 5)

# Now Split the countvectors and target (y)
splits = skf.split(data_1, y)

# iterate through the train and valid index in splits for 5 folds
for i, (train_index,test_index) in enumerate(splits):

  # Get X_train, X_valid, y_train, y_valid using indexes

    tf.keras.backend.clear_session()

    print("For fold : {}".format(i+1))
    # print("Index train : {}".format(train_index))
    # print("Index tesst : {}".format(test_index))

    X_train, X_valid = data_1[train_index] , data_1[test_index]
    y_train, y_valid = y[train_index] , y[test_index]

    #call the build_model function and initialize the model
    model = build_model(X_train)

    # train and validate the model on the count vectors of text which we have
    # adjust batch size according to your computation power (suggestion use :
    history = model.fit(X_train, y_train, epochs = 5, batch_size = 16 , valida

    # plot the graph between training auc and validation auc
    fig,fig_2 = plt.subplots(1, 2, figsize = (10,5))

    fig_2[0].plot(history.history['loss'])
    fig_2[0].plot(history.history['val_loss'])
    fig_2[0].set_title('Model Loss')
    fig_2[0].set(xlabel = 'Epoch', ylabel = 'Loss')
    fig_2[0].legend(['Train', 'Val'], loc = 'upper left')

    fig_2[1].plot(history.history['auc'])
    fig_2[1].plot(history.history['val_auc'])
    fig_2[1].set_title('Model AUC')
    fig_2[1].set(xlabel = 'Epoch', ylabel = 'AUC')
    fig_2[1].legend(['Train', 'Val'], loc = 'upper left')

    plt.show()
```

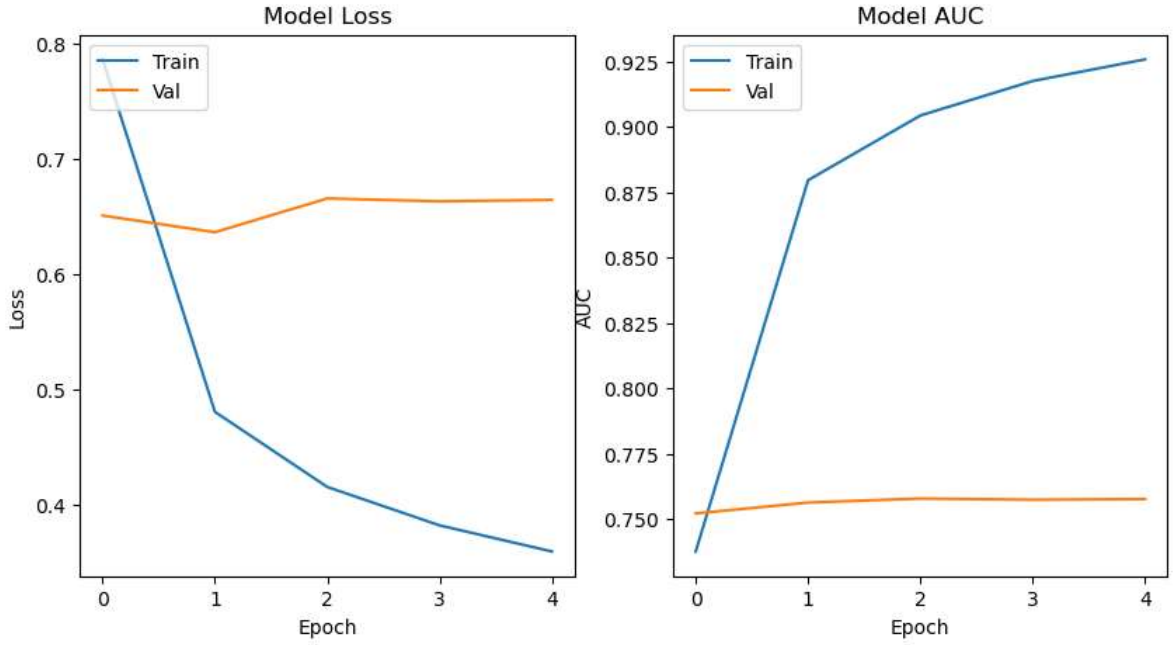
```
For fold : 1
Epoch 1/5
381/381 [==============================] - 5s 12ms/step - loss: 0.7821 - auc:
0.7380 - val_loss: 0.6459 - val_auc: 0.7345
Epoch 2/5
381/381 [==============================] - 5s 12ms/step - loss: 0.4949 - auc:
0.8728 - val_loss: 0.6312 - val_auc: 0.7364
Epoch 3/5
381/381 [==============================] - 5s 13ms/step - loss: 0.4275 - auc:
0.8993 - val_loss: 0.6354 - val_auc: 0.7411
Epoch 4/5
381/381 [==============================] - 4s 11ms/step - loss: 0.3980 - auc:
0.9118 - val_loss: 0.6275 - val_auc: 0.7417
Epoch 5/5
381/381 [==============================] - 4s 11ms/step - loss: 0.3774 - auc:
0.9191 - val_loss: 0.6254 - val_auc: 0.7397
```



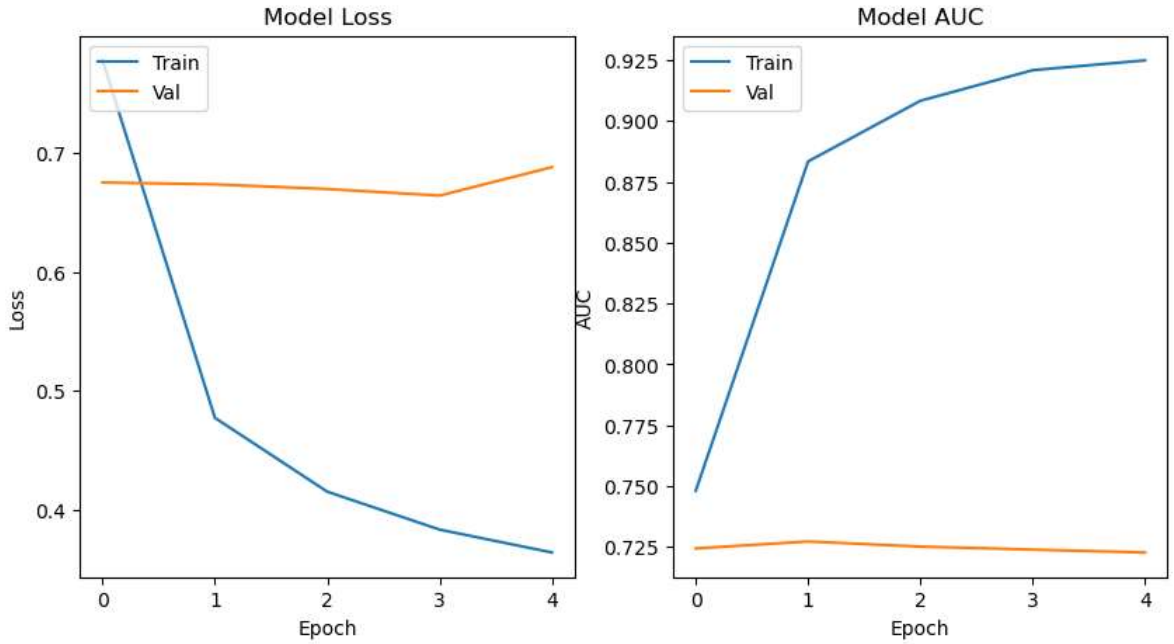

```
For fold : 2
Epoch 1/5
381/381 [==============================] - 5s 12ms/step - loss: 0.7680 - auc:
0.7506 - val_loss: 0.6645 - val_auc: 0.7110
Epoch 2/5
381/381 [==============================] - 4s 11ms/step - loss: 0.4780 - auc:
0.8849 - val_loss: 0.6881 - val_auc: 0.7070
Epoch 3/5
381/381 [==============================] - 4s 11ms/step - loss: 0.4147 - auc:
0.9083 - val_loss: 0.7187 - val_auc: 0.7051
Epoch 4/5
381/381 [==============================] - 4s 11ms/step - loss: 0.3826 - auc:
0.9194 - val_loss: 0.7436 - val_auc: 0.6987
Epoch 5/5
381/381 [==============================] - 4s 11ms/step - loss: 0.3581 - auc:
0.9281 - val_loss: 0.7476 - val_auc: 0.6958
```



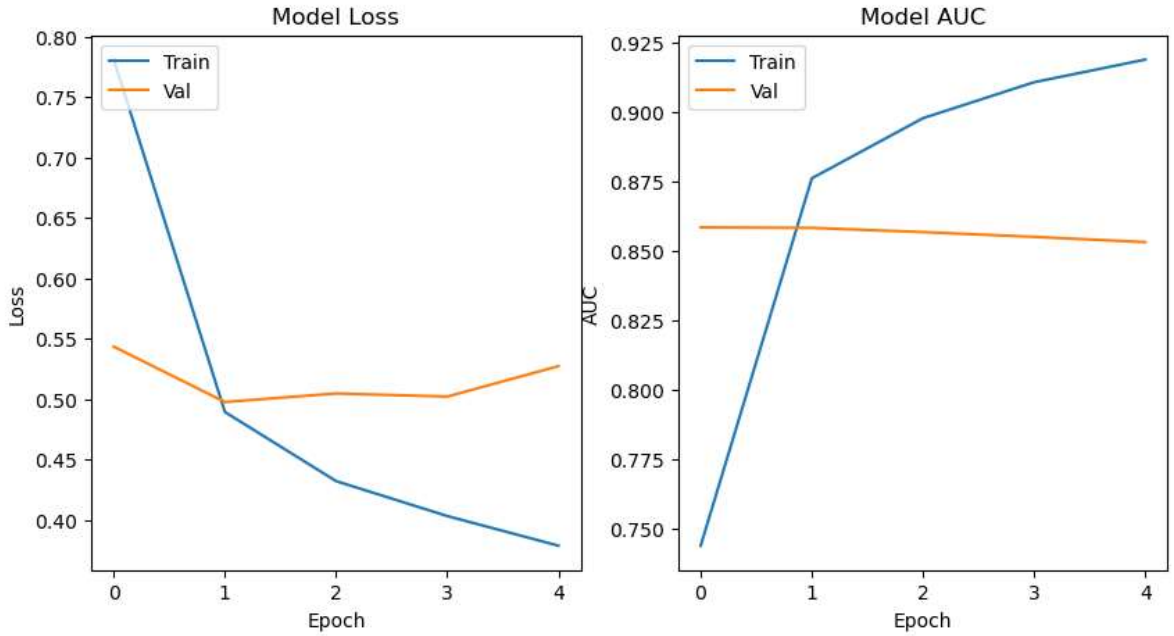

```
For fold : 3
Epoch 1/5
381/381 [==============================] - 4s 11ms/step - loss: 0.7863 - auc:
0.7377 - val_loss: 0.6507 - val_auc: 0.7522
Epoch 2/5
381/381 [==============================] - 4s 11ms/step - loss: 0.4802 - auc:
0.8796 - val_loss: 0.6362 - val_auc: 0.7564
Epoch 3/5
381/381 [==============================] - 4s 12ms/step - loss: 0.4150 - auc:
0.9044 - val_loss: 0.6655 - val_auc: 0.7579
Epoch 4/5
381/381 [==============================] - 5s 12ms/step - loss: 0.3817 - auc:
0.9175 - val_loss: 0.6630 - val_auc: 0.7574
Epoch 5/5
381/381 [==============================] - 4s 11ms/step - loss: 0.3590 - auc:
0.9258 - val_loss: 0.6643 - val_auc: 0.7577
```



```
For fold : 4
Epoch 1/5
381/381 [==============================] - 5s 11ms/step - loss: 0.7779 - auc:
0.7480 - val_loss: 0.6751 - val_auc: 0.7243
Epoch 2/5
381/381 [==============================] - 4s 11ms/step - loss: 0.4770 - auc:
0.8834 - val_loss: 0.6736 - val_auc: 0.7272
Epoch 3/5
381/381 [==============================] - 4s 12ms/step - loss: 0.4149 - auc:
0.9083 - val_loss: 0.6697 - val_auc: 0.7251
Epoch 4/5
381/381 [==============================] - 4s 11ms/step - loss: 0.3830 - auc:
0.9209 - val_loss: 0.6641 - val_auc: 0.7238
Epoch 5/5
381/381 [==============================] - 4s 12ms/step - loss: 0.3637 - auc:
0.9249 - val_loss: 0.6880 - val_auc: 0.7226
```




```
For fold : 5
Epoch 1/5
381/381 [==============================] - 5s 11ms/step - loss: 0.7808 - auc:
0.7438 - val_loss: 0.5432 - val_auc: 0.8585
Epoch 2/5
381/381 [==============================] - 4s 11ms/step - loss: 0.4892 - auc:
0.8762 - val_loss: 0.4974 - val_auc: 0.8583
Epoch 3/5
381/381 [==============================] - 4s 11ms/step - loss: 0.4320 - auc:
0.8978 - val_loss: 0.5045 - val_auc: 0.8568
Epoch 4/5
381/381 [==============================] - 4s 12ms/step - loss: 0.4031 - auc:
0.9108 - val_loss: 0.5020 - val_auc: 0.8551
Epoch 5/5
381/381 [==============================] - 5s 12ms/step - loss: 0.3785 - auc:
0.9190 - val_loss: 0.5271 - val_auc: 0.8532
```



Q) Comment on the plots. How did it varied across different folds for neural networks?

Observation:

For the graphs plotted on the left, we can see a rapid decrease in the training loss with increasing number of epochs where as with validation loss we can only see a slight decrease which increasing number of epochs.

For the graphs plotted on the right, we can a an increase in the training accuracy. For validation accuracy we can see that sharp increase in the accuracy.

We can say that our model is not performing that well on the validation set as the validation losses don't seem to go down alongside the training losses after every epoch.

Building Multinomial Navie Bayes on Countvectors

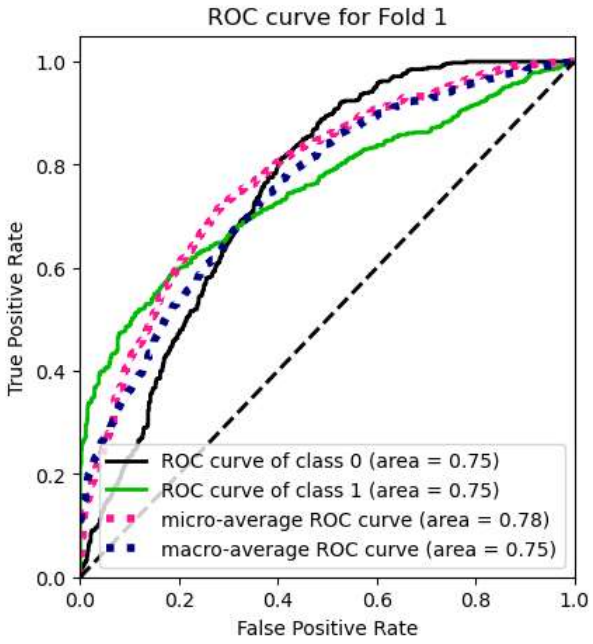
In [33]:
```
splits
```

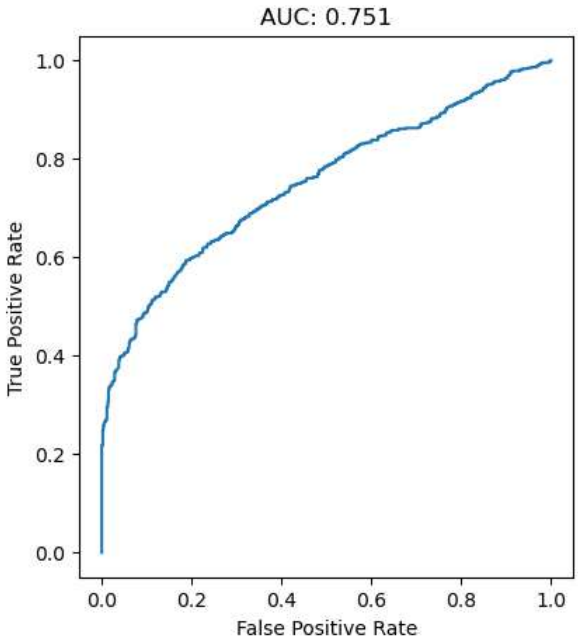
Out[33]:
```
<generator object _BaseKFold.split at 0x000001CC408F20B0>
```

In [34]:
```
skf = StratifiedKFold(n_splits = 5)

# Now Split the countvectors and target (y)
splits = skf.split(data_1, y)
```

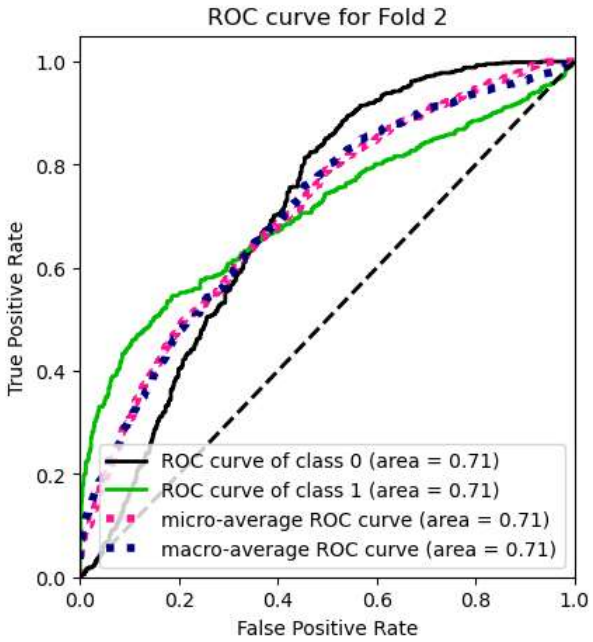
In [35]:
```
import scikitplot as skplt
from sklearn.metrics import auc
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pyplot as plt

for i, (train_index, test_index) in enumerate(splits):

    # Get X_train, X_valid, y_train, y_valid using indexes
    print("For fold : {}".format(i+1))
    X_train, X_valid = data_1[train_index], data_1[test_index]
    y_train, y_valid = y[train_index], y[test_index]

    # intialise multinomial navie bayes with default parameters
    model = MultinomialNB()

    # fit the data
    model.fit(X_train, y_train)

    # compute ROC curve for validation data
    y_scores = model.predict_proba(X_valid)[:, 1]
    fpr, tpr, _ = roc_curve(y_valid, y_scores)

    # compute AUC score for validation data
    roc_auc = auc(fpr, tpr)

    # plot ROC curve and AUC score for validation data
    fig, ax = plt.subplots(1, 2, figsize=(10, 5))
    skplt.metrics.plot_roc(y_valid, model.predict_proba(X_valid), ax=ax[0])
    ax[0].set_title(f"ROC curve for Fold {i+1}")
    ax[1].plot(fpr, tpr)
    ax[1].set_xlabel("False Positive Rate")
    ax[1].set_ylabel("True Positive Rate")
    ax[1].set_title(f"AUC: {roc_auc:.3f}")
    plt.show()
```

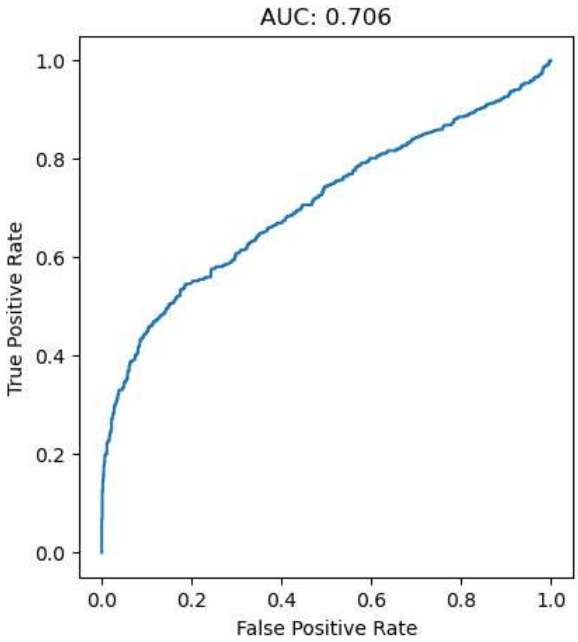
```
For fold : 1
```



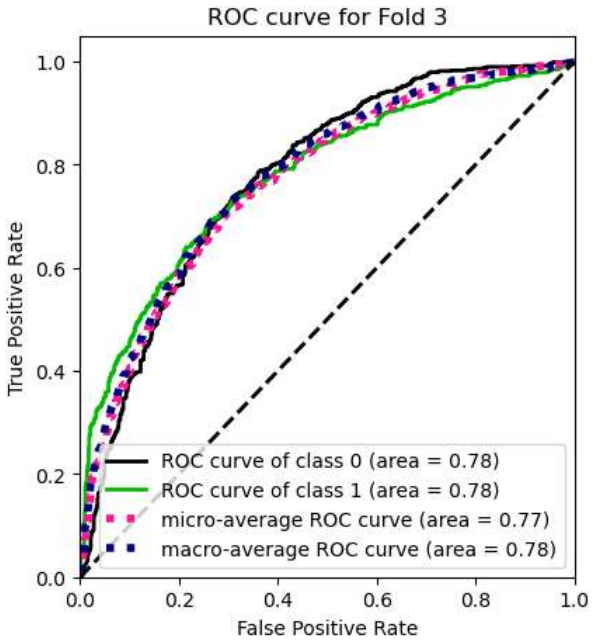


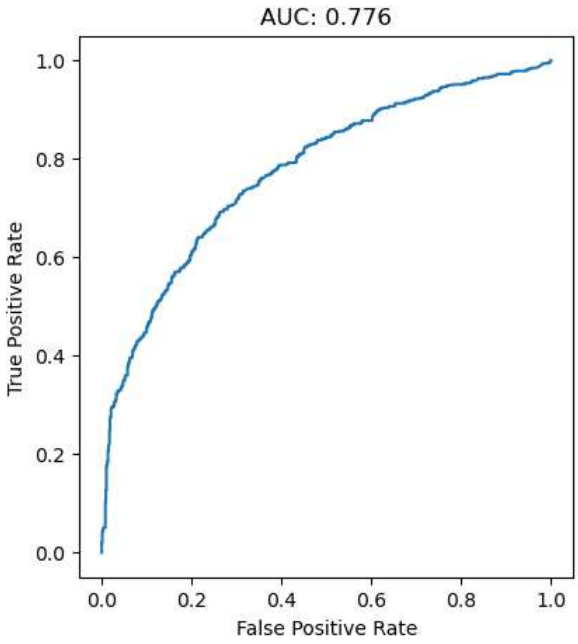
For fold : 2



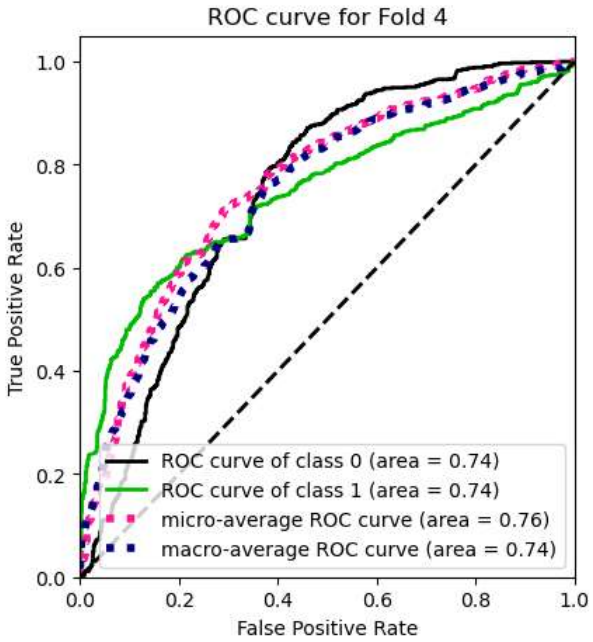


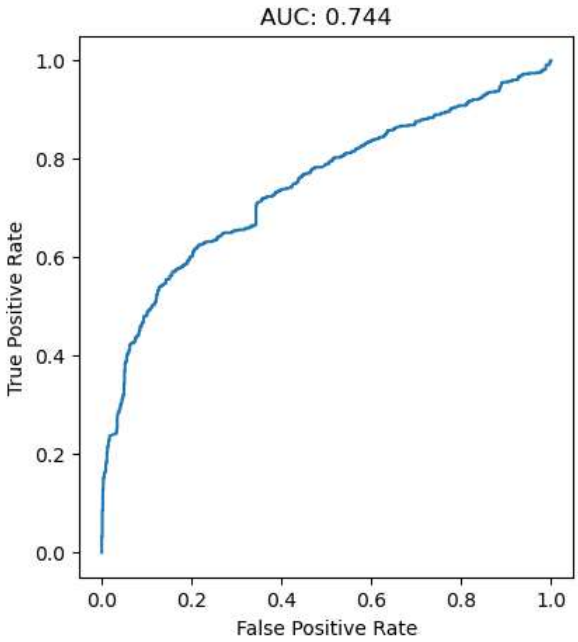
For fold : 3



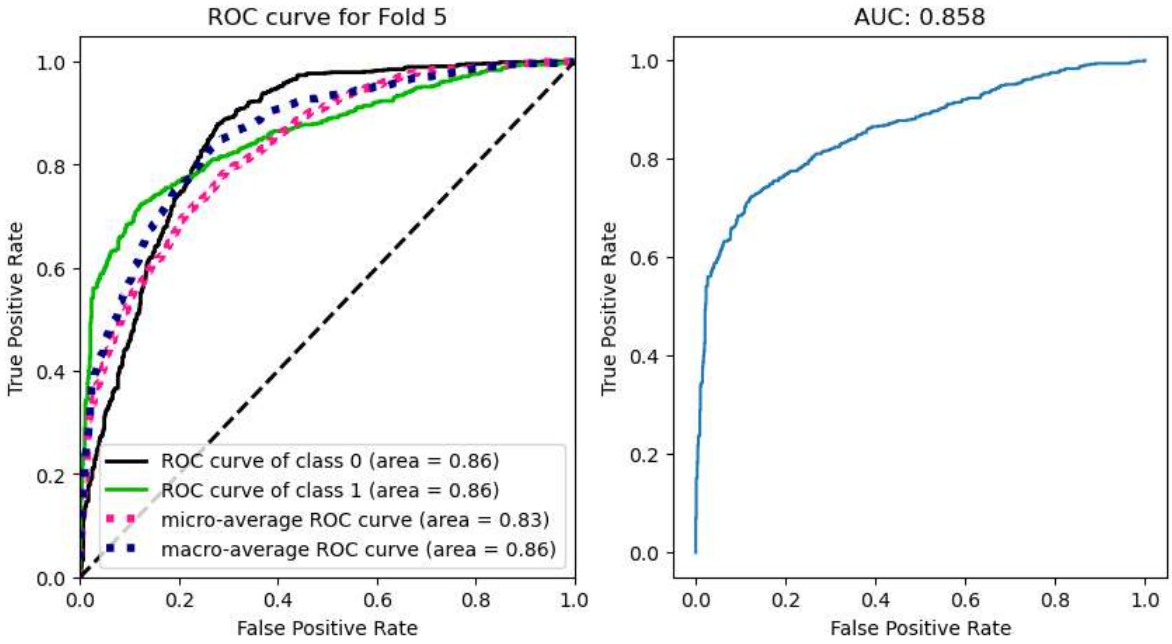


For fold : 4





For fold : 5



Q) Comment on the plots. How did it varied across different folds for Multinomial Naive Bayes?

Observation:

The AUC for the plots between False Positive Rate and True Positive Rate is fluctuating, the AUC started with 0.751 for the 1st Fold and increased with the final fold that is for the 5th, the AUC is 0.858.

The ROC for the plots between False Positive Rate and True Positive Rate is fluctuating, the ROC started with 0.78 for the 1st Fold and increased with the final fold that is for the 5th, the ROC is 0.86. The ROC value falls and increases consecutively for folds.

Question 4 Theory Question

What is the difference between Count Vectorizer, TFIDF, Word2Vec and Glove? (5 points)

Answer:

1. Count Vectorizer is a text feature extraction method that belongs to the common class of feature value calculation. For each training text, it only considers the frequency of each word in that training text. countVectorizer converts the words in the text into a word frequency matrix, and it calculates the number of occurrences of each word by the fit_transform function.

2. TF-IDF is a statistical method to evaluate the importance of a word for a document set or one of the documents in a corpus. The importance of a word increases proportionally with its occurrence in a document, but decreases inversely with its frequency in a corpus. various forms of TF-IDF weighting are often applied by search engines as a measure or rating of the relevance between a document and a user's query. the main idea of TF-IDF is

that if a word or phrase has a high frequency of TF occurrence in an article and a low frequency of TF occurrence in other articles, then the word or phrase is considered to have good category differentiation ability and is suitable for classification.

3. Word2vec is a group of related models used to generate word vectors. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic word text. The networks are represented by words and need to guess the input words in adjacent positions, under the assumption of the bag-of-words model in word2vec, the order of words is not important. After training, the word2vec model can be used to map each word to a vector, which can be used to represent word-to-word relationships, and is the hidden layer of the neural network. It is the process of transforming words into "computable", "structured" vectors.

4. Glove is a word representation tool based on global word frequency statistics, it can list a word into a vector of real numbers and calculate the semantic similarity between two words by operations on the vector, such as Euclidean distance or cosine similarity.

The differences between them are:

1. Glove and Word2vec are both unsupervised models for generating word vectors. Both models can encode words into a vector based on the "co-occurrence" information of the words (i.e. the frequency of occurrence of a word in the corpus). The difference lies in the mechanism used to generate the word vectors, with word2vec being a "predictive" model and GloVe being a "count-based" model. The word vectors generated by either of these models can be used for a variety of tasks.

2. For each training text, CountVectorizer only considers the frequency of each word in the training text, while TfidfVectorizer considers the frequency of a word in the current training text and the inverse of the number of other training texts containing this word. In contrast, the larger the number of training texts, the more advantageous the feature quantification approach of TfidfVectorizer is.

3. TF-IDF is a word-document mapping (with some normalization). It ignores the order of words and gives an nxm matrix (or mxn, depending on the implementation), where n is the number of words in the vocabulary and m is the number of documents. On the other hand, Word2Vec provides a unique vector for each word, based on the words that appear around that particular word.TF-IDF is obtained from simple linear algebra.Word2Vec is obtained from the hidden layer of a two-layer neural network.TF-IDF can be used to assign vectors to either words or documents.Word2Vec can be used directly to assign a vector to a word, but to Word2Vec can be used directly to assign a vector to a word, but to obtain a vector representation of a document requires further processing. Unlike TF-IDF, Word2Vec takes into account the position of the word in the document.

What is the significant difference between the Niave Bayes Implementation using Bag of Words and TF-IDF? (5 points)

Answer:

1. Bag of Words just creates a set of vectors containing the count of word occurrences in the document (reviews), while the TF-IDF model contains information on the more important words and the less important ones as well.

2. Bag of Words: This model ignores the grammatical and sequential elements of the text and considers it as just a collection of several words, with each word occurring independently in the document. It does not consider the order of words in a sentence, but only the number of occurrences of words in the vocabulary in that sentence. The value of each position in the vector is the number of occurrences of the word corresponding to that code in the passage. In most cases, we use the bag-of-words model. Most of the text will use only a small fraction of the words in the vocabulary, so our word vector will have a large number of zeros, which means that the word vector is sparse. In practical applications sparse matrices are generally used for storage.

3. The BOW model also has many drawbacks, firstly it does not consider the order between words, and secondly it does not reflect the keywords of a sentence. The bag-of-words model considers that the words with more occurrences in the text have more weight, so the value is the number of times the word appears in the text.

4. The TF-IDF model has the same idea as the bag-of-words model, except that the value of the vector is different. TF-IDF thinks that words like "the", "I" and "you" must appear more often in the text, but it really doesn't make sense. TF-IDF introduces document frequency to weaken the weight of words like "the", "I", and "you". Document frequency is the number of times a word appears in a document. TF refers to the number of occurrences of a word in the text in the bag-of-words model. The larger the TF-IDF of a word in an article, the higher the importance of the word in the article, so the TF-IDF of each word in the article is calculated and sorted from largest to smallest, and the words at the top are the keywords of the article.

5. Although TF-IDF has the advantages of being simple, fast, and easy to understand. However, it also has disadvantages: sometimes using word frequency to measure the importance of a word in an article is not comprehensive enough, sometimes important words may not appear enough, and this calculation cannot reflect positional information and the importance of words in context.

In [ ]: