CS 584-A: Natural Language Processing

Student information

- Full name: Thanapoom Phatthanaphan
- CWID: 20011296

Homework 1

Goals

The goal of HW1 is for you to get familiar with extracting features and standard machine learning workflows, including reading data, preprocessing data, training, testing, and evaluation. All those are open questions and there is no fixed solution. The difference in data processing, parameter initialization, data split, etc, will lead to the differences in final predications and evaluation results. Therefore, during the grading, the specific values in the results are not important. It is important that you focus on implementing the functions and setting up the pipelines.

Dataset

The dataset is about the product review dataset from Amazon. Please download it from Canvas. The dataset provides product reviews and overall ratings for 4,195 products. We will consider the fine-grained overall ratings as labels, which are ranging from 1 to 5: highly negative, negative, neutral, positive, and highly positive.

Task: Sentiment Abalysis with Text Classification

Based on the dataset, the task is to perform sentiment analysis, which predicts the sentiments based on the free-text reviews. You can solve the task as a multi-class classification problem (5 classes in total). You can also simplify the task as a binary classification problem. Specifically, we will consider the rating 4 and 5 as positive, and rating 1 and 2 as negative. For data samples with neutral rating 3, you can discard them or simply consider them as either positive or negative samples.

In the submission, please make sure to clearly mention which task you are solving and how did you prepare the class labels.

Task 1: Extraction features

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
ll drive.mount("/content/drive", force_remount=True).
```

1. Data preparation

In [2]:
```
# Import necessary libraries for data preparation
import numpy as np
import pandas as pd
import nltk
import string
from nltk.corpus import stopwords
import re
```

1.1 Data preprocessing

Download and load the dataset. Please process the reviews by
1) converting all text to lowercase to ensure uniformity.
2) removing punctuations, numbers, and stopwords.
3) tokenizing the reviews into tokens. If you plan to work on the binary classification problem, you will need to assign binary class labels based on the above-mentioned strategy.

In [3]:
```
# Import dataset
dataset = pd.read_csv("/content/drive/My Drive/Colab Notebooks/amazon_review
dataset
```

Out[3]:

	overall	reviewText
0	4	No issues.
1	5	Purchased this for my device, it worked as adv...
2	4	it works as expected. I should have sprung for...
3	5	This think has worked out great.Had a diff. br...
4	5	Bought it with Retail Packaging, arrived legit...
...	...	...
4910	1	I bought this Sandisk 16GB Class 10 to use wit...
4911	5	Used this for extending the capabilities of my...
4912	5	Great card that is very fast and reliable. It ...
4913	5	Good amount of space for the stuff I want to d...
4914	5	I've heard bad things about this 64gb Micro SD...

4915 rows × 2 columns

In [4]:
```
def clean_text(text):

    """ A function to clean the text"""

    # Convert all text to lowercase to ensure uniformity
    text = str(text).lower()

    # Remove punctuations, and numbers
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    text = re.sub(r'\d', '', text)

    # Create a set of stopwords
```

```
    stop_words = set(stopwords.words('english'))

    # Tokenize the text into words
    words = tokenization(text)

    tokenized_text = []
    for token in words:

        # Get the token excludes those stopwords
        if token not in stop_words:
            tokenized_text.append(token)

    return tokenized_text

def tokenization(text):

    """ A function to tokenize the text """

    words = nltk.word_tokenize(text)

    return words

dataset["reviewText"] = dataset["reviewText"].apply(clean_text)
dataset
```

Out[4]:

	overall	reviewText
0	4	[issues]
1	5	[purchased, device, worked, advertised, never,...
2	4	[works, expected, sprung, higher, capacity, th...
3	5	[think, worked, greathad, diff, bran, gb, card...
4	5	[bought, retail, packaging, arrived, legit, or...
...	...	...
4910	1	[bought, sandisk, gb, class, use, htc, inspire...
4911	5	[used, extending, capabilities, samsung, galax...
4912	5	[great, card, fast, reliable, comes, optional,...
4913	5	[good, amount, space, stuff, want, fits, gopro...
4914	5	[ive, heard, bad, things, gb, micro, sd, card,...

4915 rows × 2 columns

1.2 Data split

Split the data with the ratio of 0.8, 0.1, and 0.1 into training, validation/development, and testing, respectively.

In [5]:
```
# Import necessary library for splitting data
from sklearn.model_selection import train_test_split
```

In [6]:
```
def split_data(dataset, train_data_ratio, validate_data_ratio, test_data_rat

    """A function to split the dataset into 3 parts including,
    train data, validation data, and test data"""
```

```
    train_data, remaining_data = train_test_split(dataset,
                                                  train_size=0.8,
                                                  random_state=100)
    validate_data, test_data = train_test_split(remaining_data,
                                                test_size=0.1,
                                                random_state=100)
    splitted_data = [train_data, validate_data, test_data]

    return splitted_data

# Define the ratio for train data, validation data, and test data
train_data_ratio, validate_data_ratio, test_data_ratio = 0.8, 0.1, 0.1

# Split the dataset into 3 parts including, train data, validation data, and
splitted_data = split_data(dataset, train_data_ratio, validate_data_ratio, t
train_data, validate_data, test_data = splitted_data[0], splitted_data[1], s
```

1.3 Data statistics

Calculate the basic statistics of the splitted data (train data, validate data, test data) including, the number of samples, the minimum, maximum, and average number of tokens, the number of positive, and negative reviews.

In [7]:

```
# Get the number of tokens
def get_num_tokens(tokens_list):

    """A function to get the number of tokens"""

    return len(tokens_list)

# Calculate basic statistics of the data
dataset_info = {
    'Group of data': ['Train data',
                      'Validate data',
                      'Test data'],
    '#data samples': [len(train_data),
                      len(validate_data),
                      len(test_data)],
    'Min. #tokens': [min(train_data['reviewText'].apply(get_num_tokens)),
                     min(validate_data['reviewText'].apply(get_num_tokens)),
                     min(test_data['reviewText'].apply(get_num_tokens))],
    'Min. #tokens of positive reviews': [min(train_data['reviewText'][train_
                                         min(validate_data['reviewText'][val
                                         min(test_data['reviewText'][test_da
    'Min. #tokens of neutral reviews': [min(train_data['reviewText'][train_c
                                       min(validate_data['reviewText'][vali
                                       min(test_data['reviewText'][test_dat
    'Min. #tokens of negative reviews': [min(train_data['reviewText'][train_
                                         min(validate_data['reviewText'][val
                                         min(test_data['reviewText'][test_da
    'Avg. #tokens': [train_data['reviewText'].apply(get_num_tokens).mean(),
                     validate_data['reviewText'].apply(get_num_tokens).mean(
                     test_data['reviewText'].apply(get_num_tokens).mean()],
    'Avg. #tokens of positive reviews': [train_data['reviewText'][train_data
                                         validate_data['reviewText'][validat
                                         test_data['reviewText'][test_data['
    'Avg. #tokens of neutral reviews': [train_data['reviewText'][train_data[
                                       validate_data['reviewText'][validate
                                       test_data['reviewText'][test_data['c
    'Avg. #tokens of negative reviews': [train_data['reviewText'][train_data
                                         validate_data['reviewText'][validat
                                         test_data['reviewText'][test_data['
```

```
    'Max. #tokens': [max(train_data['reviewText'].apply(get_num_tokens)),
                     max(validate_data['reviewText'].apply(get_num_tokens)),
                     max(test_data['reviewText'].apply(get_num_tokens))],
    'Max. #tokens of positive reviews': [max(train_data['reviewText'][train_
                                         max(validate_data['reviewText'][val
                                         max(test_data['reviewText'][test_da
    'Max. #tokens of neutral reviews': [max(train_data['reviewText'][train_c
                                       max(validate_data['reviewText'][vali
                                       max(test_data['reviewText'][test_dat
    'Max. #tokens of negative reviews': [max(train_data['reviewText'][train_
                                         max(validate_data['reviewText'][val
                                         max(test_data['reviewText'][test_da
    '#positive reviews': [len(train_data[train_data['overall'] >= 4]),
                          len(validate_data[validate_data['overall'] >= 4]),
                          len(test_data[test_data['overall'] >= 4])],
    '#neutral reviews': [len(train_data[train_data['overall'] == 3]),
                         len(validate_data[validate_data['overall'] == 3]),
                         len(test_data[test_data['overall'] == 3])],
    '#negative reviews': [len(train_data[train_data['overall'] < 3]),
                          len(validate_data[validate_data['overall'] < 3]),
                          len(test_data[test_data['overall'] < 3])]
}

# Create a DataFrame that contains the statistics of the data
dataset_info_table = pd.DataFrame(dataset_info)

# Print the statistics table
dataset_info_table
```

Out[7]:

	Group of data	#data samples	Min. #tokens	Min. #tokens of positive reviews	Min. #tokens of neutral reviews	Min. #tokens of negative reviews	Avg. #tokens	Avg. #tokens of positive reviews	Avg #token of neutra review
0	Train data	3932	1	1	4	2	25.332655	22.894722	34.25454
1	Validate data	884	1	1	9	8	25.682127	23.543179	43.62069
2	Test data	99	6	6	18	8	24.262626	23.011364	34.00000

2. Representation of Texts: word vectors

2.1 Count-based word vectors with co-ocurrence matrix

2.1a

Please implement a function named get_vacab(corpus) that returns corpus_words, which is the list of all the distinct words used in the review corpus. You can do this with 'for' loops, but it's more efficient to do it with Python list comprehensions. The returned corpus_words should be sorted. You can use python's sorted function for this.

In [8]:

```
# Create a function to get corpus words
def get_vacab(corpus):

    """ Determine a list of distinct words for the corpus.
```

```
        Params:
            corpus (list of list of strings): corpus of documents
        Return:
            corpus_words (list of strings): sorted list of distinct words ac
    """

    # Create a blank list to store distinct words
    distinct_words = []

    # Iterate through each list in the corpus
    for i in range(len(corpus)):
        for word in corpus[i]:

            # Check that a word is not in the list of distinct words
            if word not in distinct_words:

                # Add a word to the list
                distinct_words.append(word)

    # Ascending sort the list of distinct words
    corpus_words = sorted(distinct_words)

    return corpus_words

# Call the function to get corpus words and print to see the output
corpus_words = get_vacab(dataset['reviewText'])
corpus_words
```

Out[8]:
```
['aac',
 'aas',
 'aba',
 'abdroid',
 'abilities',
 'ability',
 'able',
 'aboutgood',
 'abouti',
 'aboutherehere',
 'aboutto',
 'abovei',
 'abroad',
 'abruptly',
 'absolute',
 'absolutely',
 'abt',
 'abuse',
 'abused',
 'abysmal',
 'accdientally',
 'accept',
 'acceptable',
 'acceptably',
 'accepted',
 'accepting',
 'accepts',
 'access',
 'accessed',
 'accesses',
 'accessible',
 'accessing',
 'accessories',
 'accessory',
 'accessoryalternative',
 'accident',
 'accidentally',
 'accidently',
 'acclimated',
 'accolades',
 'accommodate',
 'accomplish',
 'accord',
 'according',
 'accordingly',
 'account',
 'accros',
 'accurate',
 'accuratei',
 'accuratethe',
 'ace',
 'acer',
 'acess',
 'acheive',
 'achieve',
 'achieved',
 'achievedusing',
 'achieves',
 'acitve',
 'acknowledge',
 'acknowledged',
 'acontourroam',
 'acquire',
 'acquired',
```

```
'acronis',
'acronyms',
'across',
'act',
'acting',
'action',
'actioncam',
'activating',
'active',
'activity',
'acts',
'actual',
'actuality',
'actually',
'ad',
'adapater',
'adapt',
'adapted',
'adapter',
'adapterbased',
'adapterbingo',
'adapterbqzhjs',
'adapterexchanging',
'adapterfunny',
'adapteri',
'adapterit',
'adapterive',
'adapterlifetime',
'adapterlol',
'adapterother',
'adapterprecautions',
'adapterpretty',
'adapters',
'adaptersamsung',
'adaptersandisk',
'adaptershipping',
'adaptersome',
'adaptersuper',
'adapterthis',
'adapterwhich',
'adapterwith',
'adapterwithout',
'adapterworks',
'adaptor',
'adaptornow',
'adaptors',
'adaptorsthat',
'adaquate',
'adata',
'adatas',
'add',
'added',
'adding',
'addingdeletingmoving',
'addition',
'additional',
'additionals',
'addon',
'address',
'addressed',
'adds',
'addsi',
'adequate',
'adequatei',
```

```
'adequately',
'adjust',
'admiral',
'admit',
'admits',
'admitted',
'admittedly',
'adn',
'adopter',
'adopting',
'adpator',
'ads',
'adult',
'advance',
'advanced',
'advancing',
'advantage',
'adventure',
'adventures',
'adventuresthis',
'adversities',
'advertise',
'advertised',
'advertisedgood',
'advertisedi',
'advertisedif',
'advertisedthis',
'advertisedwhat',
'advertisement',
'advertisements',
'advertises',
'advertising',
'advertized',
'advice',
'advise',
'advised',
'adware',
'aelago',
'aerobic',
'affect',
'affected',
'affecting',
'afford',
'affordable',
'affpa',
'afraid',
'aft',
'afternoon',
'afterward',
'afterwards',
'againalso',
'againas',
'againdecember',
'againedit',
'againi',
'againif',
'againit',
'againlolgo',
'againspeed',
'againwriting',
'age',
'agent',
'agents',
'aging',
```

```
'ago',
'agoi',
'agothankfully',
'agoworks',
'agptek',
'agree',
'agreed',
'ah',
'ahaha',
'ahead',
'aiiiighsuperspeedy',
'ainol',
'aint',
'air',
'airdroid',
'airplane',
'airport',
'aka',
'akingston',
'al',
'alamcenar',
'alaska',
'alaskai',
'albeit',
'album',
'albums',
'aldo',
'alec',
'alert',
'alerted',
'alerting',
'alerts',
'alien',
'alike',
'alla',
'allbased',
'allblack',
'alldaymalltm',
'alleged',
'allegedly',
'allexcellent',
'allgb',
'allhave',
'alli',
'allinone',
'allinstallation',
'allocated',
'allocation',
'allot',
'allotted',
'allow',
'allowed',
'allowing',
'allows',
'allpurpose',
'allready',
'allsome',
'allthough',
'almost',
'almsot',
'alone',
'alonethe',
'along',
'alot',
```

```
'already',
'alreadythis',
'alright',
'also',
'alsobuying',
'alt',
'alta',
'altered',
'alternately',
'alternative',
'although',
'altogether',
'altogetherwhen',
'altough',
'aluminum',
'alway',
'always',
'alwayspros',
'alzheimers',
'amamzon',
'amazed',
'amazementit',
'amazes',
'amazing',
'amazingi',
'amazingly',
'amazingsandisk',
'amazon',
'amazonas',
'amazoncom',
'amazonconits',
'amazonenjoy',
'amazonfrustrating',
'amazoni',
'amazonpackage',
'amazonprime',
'amazons',
'amazonseeing',
'amazonsincerelykj',
'amazonso',
'amazonthe',
'amazonthey',
'america',
'amis',
'ammount',
'amon',
'among',
'amonths',
'amount',
'amounts',
'ample',
'ampliamente',
'ana',
'anbd',
'andi',
'andiod',
'andloaded',
'andoid',
'andor',
'andriod',
'androd',
'android',
'androidread',
'androidspecific',
```

```
'ands',
'andthats',
'andy',
'anecdotal',
'anelago',
'angle',
'angryto',
'anime',
'anker',
'anniversary',
'announced',
'announcement',
'annoyance',
'annoyed',
'annoying',
'anomaly',
'another',
'anotherbigger',
'anothergreat',
'answer',
'answered',
'answers',
'answersthanks',
'anteriormente',
'anticipated',
'anticipating',
'anticipation',
'antishoplifting',
'antivirus',
'antutu',
'anyall',
'anybody',
'anycorrupted',
'anydvd',
'anyhow',
'anyjust',
'anymore',
'anymorehope',
'anymoreif',
'anymoreone',
'anyne',
'anynot',
'anyone',
'anyoneps',
'anyproblems',
'anythin',
'anything',
'anythingvery',
'anythingworks',
'anytime',
'anyway',
'anywayany',
'anyways',
'anywhere',
'anywho',
'aokp',
'aosp',
'apapter',
'apart',
'apartment',
'apex',
'aplomb',
'app',
'apparent',
```

```
'apparently',
'appbrowsernetflixmusic',
'appealed',
'appealing',
'appear',
'appearance',
'appeared',
'appearing',
'appears',
'appendage',
'apple',
'appliance',
'applicable',
'application',
'applicationas',
'applicationi',
'applications',
'applied',
'apply',
'apposed',
'appreadergb',
'appreciate',
'appreciated',
'appreciates',
'appreciative',
'apprehensive',
'approaching',
'appropriate',
'approval',
'approve',
'approved',
'approx',
'approximately',
'apps',
'appsd',
'appsgames',
'appsi',
'apr',
'april',
'ar',
'arc',
'architecture',
'archive',
'archos',
'arduous',
'area',
'areanyways',
'areas',
'arehaving',
'arei',
'arent',
'argue',
'arguing',
'arise',
'ariseit',
'arises',
'arm',
'armed',
'arose',
'around',
'aroundand',
'aroundsc',
'arounf',
'arranca',
```

```
'arrival',
'arrivalwhat',
'arrive',
'arrived',
'arrivedopting',
'arrives',
'art',
'arthritis',
'artiaga',
'articles',
'articleso',
'asamsung',
'ashtray',
'asian',
'aside',
'asis',
'asix',
'ask',
'asked',
'aski',
'asking',
'asks',
'asleep',
'asneeded',
'aspect',
'asphalt',
'aspirecarry',
'ass',
'assembly',
'assess',
'assessment',
'asset',
'assign',
'assist',
'association',
'assorted',
'assortment',
'assume',
'assumed',
'assumedprosincredible',
'assuming',
'assurance',
'assure',
'assured',
'assuringly',
'astak',
'astonishingly',
'asus',
'asusprime',
'ati',
'ativ',
'ativpc',
'atlanta',
'atlantic',
'atleast',
'atop',
'atore',
'atrecover',
'atrix',
'atrocious',
'att',
'attached',
'attaches',
'attaching',
```

```
'attachment',
'attain',
'attempt',
'attempted',
'attempting',
'attempts',
'attention',
'attest',
'atto',
'attracted',
'attractive',
'attribute',
'auction',
'audible',
'audio',
'audiobooks',
'audiovideo',
'aug',
'augmentation',
'august',
'authentic',
'authenticate',
'authenticity',
'authenticthe',
'authorized',
'auththentic',
'auto',
'automatic',
'automatically',
'automotive',
'automount',
'autophoto',
'autoscanned',
'av',
'avail',
'available',
'availablethe',
'average',
'averaged',
'averages',
'avg',
'avoid',
'avoided',
'avoiding',
'await',
'awaiting',
'aware',
'awareif',
'away',
'awaygb',
'awayroot',
'awe',
'awesome',
'awesomeconsnot',
'awesomei',
'awesomely',
'awhile',
'awry',
'ax',
'axiom',
'axxholes',
'ayyy',
'az',
'b',
```

```
'babies',
'baby',
'babythe',
'back',
'backdecember',
'backed',
'backedup',
'backfree',
'background',
'backgroundalso',
'backing',
'backorder',
'backordered',
'backs',
'backside',
'backstill',
'backthis',
'backtoback',
'backup',
'backups',
'backward',
'backwards',
'bad',
'bada',
'badboy',
'badboys',
'badjust',
'badly',
'badupdate',
'baffled',
'bag',
'balance',
'ball',
'ballpark',
'bam',
'bandwagon',
'bandwidth',
'bang',
'bango',
'banking',
'banner',
'bar',
'bare',
'barely',
'bargain',
'bargainbasement',
'bargan',
'barn',
'base',
'based',
'basic',
'basically',
'basics',
'basis',
'basketball',
'bass',
'bast',
'basta',
'bat',
'batch',
'batches',
'batchgopro',
'batchive',
'batteries',
```

```
'battery',
'bay',
'bays',
'bb',
'bbboth',
'bconvience',
'bcuz',
'beagle',
'bean',
'bear',
'bearing',
'beast',
'beat',
'beatifully',
'beating',
'beats',
'beautiful',
'beautifully',
'beauty',
'became',
'becasue',
'becauseive',
'become',
'becomes',
'becoming',
'becuase',
'becuse',
'bed',
'beef',
'beefy',
'beforecon',
'beforehand',
'beforemicrosd',
'beforeso',
'beg',
'began',
'begin',
'beginning',
'begins',
'begun',
'behave',
'behaved',
'behaves',
'behavior',
'behavioral',
'behind',
'behold',
'beijing',
'beingit',
'beit',
'beleive',
'believ',
'believe',
'believer',
'beloved',
'belowfirstly',
'bemuch',
'bench',
'benchmark',
'benchmarked',
'benchmarking',
'benchmarks',
'benchmarksettings',
'benchmarkspurchased',
```

```
'bend',
'bene',
'benefit',
'benefits',
'benissimo',
'beside',
'besides',
'best',
'bestbuy',
'besti',
'bestthen',
'bet',
'better',
'betteri',
'betterthanks',
'betterwith',
'beware',
'bewareupdate',
'bext',
'beyond',
'beyondhave',
'bg',
'biased',
'bible',
'bid',
'bienahora',
'big',
'bigbox',
'bigger',
'biggerolder',
'biggest',
'biggestcapacity',
'biggie',
'bike',
'bill',
'billing',
'billion',
'bin',
'binary',
'bindings',
'bingo',
'bins',
'bionic',
'bionici',
'birthday',
'bit',
'bites',
'bitlocker',
'bitrate',
'bitrates',
'bits',
'bitty',
'bizillion',
'bla',
'black',
'blackberry',
'blackbox',
'blackfriday',
'blackno',
'blackto',
'blackvue',
'blackwith',
'blade',
'blah',
```

```
'blame',
'blamed',
'blank',
'blankit',
'blankunformatted',
'blazaing',
'blaze',
'blazes',
'blazing',
'bleeding',
'blend',
'blew',
'blink',
'blinked',
'blinks',
'blip',
'blipped',
'blips',
'bliss',
'blissful',
'blister',
'bloated',
'bloatware',
'block',
'blocked',
'blocking',
'blocks',
'blocky',
'blogs',
'blow',
'blowing',
'blown',
'blows',
'blue',
'blueand',
'bluetooth',
'bluetoothcompatible',
'bluray',
'blurays',
'bn',
'bnever',
'board',
'boards',
'boat',
'bobs',
'body',
'bogged',
'bogs',
'bogus',
'bomb',
'bone',
'bonestock',
'bonus',
'bonusnow',
'book',
'booklet',
'bookmarks',
'books',
'bookshelf',
'boost',
'boosted',
'boosts',
'boostwith',
'boot',
```

```
'bootable',
'booted',
'bootfor',
'booting',
'bootleg',
'bootlegged',
'bootloader',
'boots',
'bootso',
'bootthis',
'bootup',
'bored',
'boredom',
'boring',
'born',
'borrowed',
'bother',
'bothered',
'bothernow',
'bothso',
'boththis',
'bottleneck',
'bottlenecked',
'bottlenecking',
'bottlenecks',
'bottom',
'bough',
'bought',
'bougth',
'bounce',
'bounced',
'bouncing',
'boundary',
'bout',
'bouth',
'bouthg',
'boverall',
'box',
'boxand',
'boxes',
'boxno',
'boxsky',
'boxthe',
'boy',
'boyfriend',
'brag',
'bragged',
'brain',
'brainer',
'brainernote',
'brainerthe',
'bran',
'brand',
'brandaaaaaaaa',
'brandagain',
'branded',
'brandi',
'branding',
'brandname',
'brands',
'brandsi',
'brandversion',
'brandyou',
'bravo',
```

```
'bread',
'break',
'breakdown',
'breaker',
'breaking',
'breaks',
'breathe',
'breathing',
'breed',
'breeze',
'brew',
'brick',
'bricked',
'bridge',
'brief',
'briefly',
'bright',
'brighter',
'brilliant',
'brilliantly',
'bring',
'brings',
'brisk',
'brittle',
'broadcasts',
'broke',
'broken',
'brokenalso',
'brokencorruptunusable',
'brotha',
'brother',
'brothers',
'brought',
'brown',
'browse',
'browser',
'browsing',
'bruce',
'brutally',
'bs',
'bsi',
'btw',
'btwshame',
'bublbe',
'buck',
'bucks',
'budget',
'budgetso',
'buena',
'bueno',
'buff',
'buffer',
'buffering',
'bufferinghiccupspauses',
'buffers',
'bufferthere',
'bug',
'bugger',
'buggercons',
'buggy',
'bugs',
'build',
'building',
'buildings',
```

```
 'built',
 'builtin',
 'buit',
 'bujgger',
 'bulit',
 'bulk',
 'bullcrap',
 'bulletproof',
 'bum',
 'bummed',
 'bummer',
 'bump',
 'bumps',
 'bunch',
 'bunches',
 'bundle',
 'bundling',
 'bunk',
 'bur',
 'burchase',
 'burn',
 'burned',
 'burner',
 'burning',
 'burst',
 'bus',
 'business',
 'bust',
 'busted',
 'busy',
 'butsmall',
 'butt',
 'butter',
 'butthis',
 'button',
 'butwhat',
 'buy',
 'buyer',
 'buyers',
 'buyi',
 ...]
```

2.1b

Based on the word vocabulary obtained with get_vacab(corpus) function, please implement a function named compute_co_occurrence_matrix(corpus, window_size=4) that returns both M and word2index. Here, M is the co-occurrence matrix of word counts and word2index is a dictionary that maps word to index. The function constructs a co-occurrence matrix for a certain window-size n (with a default of 4), considering words n before and n after the word in the center of the window. You can use numpy to represent vectors, matrices, and tensors.

In [9]:
```
def compute_co_occurrence_matrix(dataset, corpus_words, window_size=4):

    """ Compute co-occurrence matrix for the given corpus and window_size (c
        Params:
            dataset (dataframe): the original dataset of reviews
            corpus_words (list of strings): the list of strings that contair
            window_size (int): size of context window
        Return:
            M (a symmetric numpy matrix of shape (number of unique words in
                Co-occurence matrix of word counts.
```

```
                The ordering of the words in the rows/columns should be the
            word2index (dict): dictionary that maps word to index (i.e. row/
    """

    # Initialize the co-occurence matrix M with zeros
    M = np.zeros((len(corpus_words), len(corpus_words)))

    # Create a hashmap to map the relationship between word and index (word
    word2index = {}
    for index, word in enumerate(corpus_words):
        word2index[word] = index

    # Iterate through each list of strings in the corpus
    for words_list in dataset['reviewText']:

        # Iterate through each word in a list of strings
        for index, word in enumerate(words_list):
            # Get the index of a center word, return -1 if not found the wor
            center_word_index = word2index.get(word, -1)

            # Check if the word is found in the hashmap
            if center_word_index != -1:
                # Define the boundaries of the window
                start = max(0, index - window_size)
                end = min(len(words_list), index + window_size + 1)

                # Iterate through the words in the window
                for index_in_window in range(start, end):

                    if index_in_window == index:
                        continue
                    else:
                        # Get the word of the current index in the window
                        word_in_window = words_list[index_in_window]
                        # Get the index of the current word in the window
                        word_in_window_index = word2index.get(word_in_windo

                        # Count the co-occurrence words in the co-occurrence
                        M[center_word_index][word_in_window_index] += 1

    return M, word2index

# Print to check the output from compute_co_occurrence_matrix function
M_count_based, word2index_count_based = compute_co_occurrence_matrix(dataset
```

In [10]:
```
M_count_based
```

Out[10]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [11]:
```
M_count_based.shape
```

Out[11]:
```
(9697, 9697)
```

In [12]:
```
word2index_count_based
```

Out[12]:

```
{'aac': 0,
 'aas': 1,
 'aba': 2,
 'abdroid': 3,
 'abilities': 4,
 'ability': 5,
 'able': 6,
 'aboutgood': 7,
 'abouti': 8,
 'aboutherehere': 9,
 'aboutto': 10,
 'abovei': 11,
 'abroad': 12,
 'abruptly': 13,
 'absolute': 14,
 'absolutely': 15,
 'abt': 16,
 'abuse': 17,
 'abused': 18,
 'abysmal': 19,
 'accdientally': 20,
 'accept': 21,
 'acceptable': 22,
 'acceptably': 23,
 'accepted': 24,
 'accepting': 25,
 'accepts': 26,
 'access': 27,
 'accessed': 28,
 'accesses': 29,
 'accessible': 30,
 'accessing': 31,
 'accessories': 32,
 'accessory': 33,
 'accessoryalternative': 34,
 'accident': 35,
 'accidentally': 36,
 'accidently': 37,
 'acclimated': 38,
 'accolades': 39,
 'accommodate': 40,
 'accomplish': 41,
 'accord': 42,
 'according': 43,
 'accordingly': 44,
 'account': 45,
 'accros': 46,
 'accurate': 47,
 'accuratei': 48,
 'accuratethe': 49,
 'ace': 50,
 'acer': 51,
 'acess': 52,
 'acheive': 53,
 'achieve': 54,
 'achieved': 55,
 'achievedusing': 56,
 'achieves': 57,
 'acitve': 58,
 'acknowledge': 59,
 'acknowledged': 60,
 'acontourroam': 61,
 'acquire': 62,
 'acquired': 63,
```

```
'acronis': 64,
'acronyms': 65,
'across': 66,
'act': 67,
'acting': 68,
'action': 69,
'actioncam': 70,
'activating': 71,
'active': 72,
'activity': 73,
'acts': 74,
'actual': 75,
'actuality': 76,
'actually': 77,
'ad': 78,
'adapater': 79,
'adapt': 80,
'adapted': 81,
'adapter': 82,
'adapterbased': 83,
'adapterbingo': 84,
'adapterbqzhjs': 85,
'adapterexchanging': 86,
'adapterfunny': 87,
'adapteri': 88,
'adapterit': 89,
'adapterive': 90,
'adapterlifetime': 91,
'adapterlol': 92,
'adapterother': 93,
'adapterprecautions': 94,
'adapterpretty': 95,
'adapters': 96,
'adaptersamsung': 97,
'adaptersandisk': 98,
'adaptershipping': 99,
'adaptersome': 100,
'adaptersuper': 101,
'adapterthis': 102,
'adapterwhich': 103,
'adapterwith': 104,
'adapterwithout': 105,
'adapterworks': 106,
'adaptor': 107,
'adaptornow': 108,
'adaptors': 109,
'adaptorsthat': 110,
'adaquate': 111,
'adata': 112,
'adatas': 113,
'add': 114,
'added': 115,
'adding': 116,
'addingdeletingmoving': 117,
'addition': 118,
'additional': 119,
'additionals': 120,
'addon': 121,
'address': 122,
'addressed': 123,
'adds': 124,
'addsi': 125,
'adequate': 126,
'adequatei': 127,
```

```
'adequately': 128,
'adjust': 129,
'admiral': 130,
'admit': 131,
'admits': 132,
'admitted': 133,
'admittedly': 134,
'adn': 135,
'adopter': 136,
'adopting': 137,
'adpator': 138,
'ads': 139,
'adult': 140,
'advance': 141,
'advanced': 142,
'advancing': 143,
'advantage': 144,
'adventure': 145,
'adventures': 146,
'adventuresthis': 147,
'adversities': 148,
'advertise': 149,
'advertised': 150,
'advertisedgood': 151,
'advertisedi': 152,
'advertisedif': 153,
'advertisedthis': 154,
'advertisedwhat': 155,
'advertisement': 156,
'advertisements': 157,
'advertises': 158,
'advertising': 159,
'advertized': 160,
'advice': 161,
'advise': 162,
'advised': 163,
'adware': 164,
'aelago': 165,
'aerobic': 166,
'affect': 167,
'affected': 168,
'affecting': 169,
'afford': 170,
'affordable': 171,
'affpa': 172,
'afraid': 173,
'aft': 174,
'afternoon': 175,
'afterward': 176,
'afterwards': 177,
'againalso': 178,
'againas': 179,
'againdecember': 180,
'againedit': 181,
'againi': 182,
'againif': 183,
'againit': 184,
'againlolgo': 185,
'againspeed': 186,
'againwriting': 187,
'age': 188,
'agent': 189,
'agents': 190,
'aging': 191,
```

```
'ago': 192,
'agoi': 193,
'agothankfully': 194,
'agoworks': 195,
'agptek': 196,
'agree': 197,
'agreed': 198,
'ah': 199,
'ahaha': 200,
'ahead': 201,
'aiiiighsuperspeedy': 202,
'ainol': 203,
'aint': 204,
'air': 205,
'airdroid': 206,
'airplane': 207,
'airport': 208,
'aka': 209,
'akingston': 210,
'al': 211,
'alamcenar': 212,
'alaska': 213,
'alaskai': 214,
'albeit': 215,
'album': 216,
'albums': 217,
'aldo': 218,
'alec': 219,
'alert': 220,
'alerted': 221,
'alerting': 222,
'alerts': 223,
'alien': 224,
'alike': 225,
'alla': 226,
'allbased': 227,
'allblack': 228,
'alldaymalltm': 229,
'alleged': 230,
'allegedly': 231,
'allexcellent': 232,
'allgb': 233,
'allhave': 234,
'alli': 235,
'allinone': 236,
'allinstallation': 237,
'allocated': 238,
'allocation': 239,
'allot': 240,
'allotted': 241,
'allow': 242,
'allowed': 243,
'allowing': 244,
'allows': 245,
'allpurpose': 246,
'allready': 247,
'allsome': 248,
'allthough': 249,
'almost': 250,
'almsot': 251,
'alone': 252,
'alonethe': 253,
'along': 254,
'alot': 255,
```

```
'already': 256,
'alreadythis': 257,
'alright': 258,
'also': 259,
'alsobuying': 260,
'alt': 261,
'alta': 262,
'altered': 263,
'alternately': 264,
'alternative': 265,
'although': 266,
'altogether': 267,
'altogetherwhen': 268,
'altough': 269,
'aluminum': 270,
'alway': 271,
'always': 272,
'alwayspros': 273,
'alzheimers': 274,
'amamzon': 275,
'amazed': 276,
'amazementit': 277,
'amazes': 278,
'amazing': 279,
'amazingi': 280,
'amazingly': 281,
'amazingsandisk': 282,
'amazon': 283,
'amazonas': 284,
'amazoncom': 285,
'amazonconits': 286,
'amazonenjoy': 287,
'amazonfrustrating': 288,
'amazoni': 289,
'amazonpackage': 290,
'amazonprime': 291,
'amazons': 292,
'amazonseeing': 293,
'amazonsincerelykj': 294,
'amazonso': 295,
'amazonthe': 296,
'amazonthey': 297,
'america': 298,
'amis': 299,
'ammount': 300,
'amon': 301,
'among': 302,
'amonths': 303,
'amount': 304,
'amounts': 305,
'ample': 306,
'ampliamente': 307,
'ana': 308,
'anbd': 309,
'andi': 310,
'andiod': 311,
'andloaded': 312,
'andoid': 313,
'andor': 314,
'andriod': 315,
'androd': 316,
'android': 317,
'androidread': 318,
'androidspecific': 319,
```

```
'ands': 320,
'andthats': 321,
'andy': 322,
'anecdotal': 323,
'anelago': 324,
'angle': 325,
'angryto': 326,
'anime': 327,
'anker': 328,
'anniversary': 329,
'announced': 330,
'announcement': 331,
'annoyance': 332,
'annoyed': 333,
'annoying': 334,
'anomaly': 335,
'another': 336,
'anotherbigger': 337,
'anothergreat': 338,
'answer': 339,
'answered': 340,
'answers': 341,
'answersthanks': 342,
'anteriormente': 343,
'anticipated': 344,
'anticipating': 345,
'anticipation': 346,
'antishoplifting': 347,
'antivirus': 348,
'antutu': 349,
'anyall': 350,
'anybody': 351,
'anycorrupted': 352,
'anydvd': 353,
'anyhow': 354,
'anyjust': 355,
'anymore': 356,
'anymorehope': 357,
'anymoreif': 358,
'anymoreone': 359,
'anyne': 360,
'anynot': 361,
'anyone': 362,
'anyoneps': 363,
'anyproblems': 364,
'anythin': 365,
'anything': 366,
'anythingvery': 367,
'anythingworks': 368,
'anytime': 369,
'anyway': 370,
'anywayany': 371,
'anyways': 372,
'anywhere': 373,
'anywho': 374,
'aokp': 375,
'aosp': 376,
'apapter': 377,
'apart': 378,
'apartment': 379,
'apex': 380,
'aplomb': 381,
'app': 382,
'apparent': 383,
```

```
'apparently': 384,
'appbrowsernetflixmusic': 385,
'appealed': 386,
'appealing': 387,
'appear': 388,
'appearance': 389,
'appeared': 390,
'appearing': 391,
'appears': 392,
'appendage': 393,
'apple': 394,
'appliance': 395,
'applicable': 396,
'application': 397,
'applicationas': 398,
'applicationi': 399,
'applications': 400,
'applied': 401,
'apply': 402,
'apposed': 403,
'appreadergb': 404,
'appreciate': 405,
'appreciated': 406,
'appreciates': 407,
'appreciative': 408,
'apprehensive': 409,
'approaching': 410,
'appropriate': 411,
'approval': 412,
'approve': 413,
'approved': 414,
'approx': 415,
'approximately': 416,
'apps': 417,
'appsd': 418,
'appsgames': 419,
'appsi': 420,
'apr': 421,
'april': 422,
'ar': 423,
'arc': 424,
'architecture': 425,
'archive': 426,
'archos': 427,
'arduous': 428,
'area': 429,
'areanyways': 430,
'areas': 431,
'arehaving': 432,
'arei': 433,
'arent': 434,
'argue': 435,
'arguing': 436,
'arise': 437,
'ariseit': 438,
'arises': 439,
'arm': 440,
'armed': 441,
'arose': 442,
'around': 443,
'aroundand': 444,
'aroundsc': 445,
'arounf': 446,
'arranca': 447,
```

```
'arrival': 448,
'arrivalwhat': 449,
'arrive': 450,
'arrived': 451,
'arrivedopting': 452,
'arrives': 453,
'art': 454,
'arthritis': 455,
'artiaga': 456,
'articles': 457,
'articleso': 458,
'asamsung': 459,
'ashtray': 460,
'asian': 461,
'aside': 462,
'asis': 463,
'asix': 464,
'ask': 465,
'asked': 466,
'aski': 467,
'asking': 468,
'asks': 469,
'asleep': 470,
'asneeded': 471,
'aspect': 472,
'asphalt': 473,
'aspirecarry': 474,
'ass': 475,
'assembly': 476,
'assess': 477,
'assessment': 478,
'asset': 479,
'assign': 480,
'assist': 481,
'association': 482,
'assorted': 483,
'assortment': 484,
'assume': 485,
'assumed': 486,
'assumedprosincredible': 487,
'assuming': 488,
'assurance': 489,
'assure': 490,
'assured': 491,
'assuringly': 492,
'astak': 493,
'astonishingly': 494,
'asus': 495,
'asusprime': 496,
'ati': 497,
'ativ': 498,
'ativpc': 499,
'atlanta': 500,
'atlantic': 501,
'atleast': 502,
'atop': 503,
'atore': 504,
'atrecover': 505,
'atrix': 506,
'atrocious': 507,
'att': 508,
'attached': 509,
'attaches': 510,
'attaching': 511,
```

```
'attachment': 512,
'attain': 513,
'attempt': 514,
'attempted': 515,
'attempting': 516,
'attempts': 517,
'attention': 518,
'attest': 519,
'atto': 520,
'attracted': 521,
'attractive': 522,
'attribute': 523,
'auction': 524,
'audible': 525,
'audio': 526,
'audiobooks': 527,
'audiovideo': 528,
'aug': 529,
'augmentation': 530,
'august': 531,
'authentic': 532,
'authenticate': 533,
'authenticity': 534,
'authenticthe': 535,
'authorized': 536,
'auththentic': 537,
'auto': 538,
'automatic': 539,
'automatically': 540,
'automotive': 541,
'automount': 542,
'autophoto': 543,
'autoscanned': 544,
'av': 545,
'avail': 546,
'available': 547,
'availablethe': 548,
'average': 549,
'averaged': 550,
'averages': 551,
'avg': 552,
'avoid': 553,
'avoided': 554,
'avoiding': 555,
'await': 556,
'awaiting': 557,
'aware': 558,
'awareif': 559,
'away': 560,
'awaygb': 561,
'awayroot': 562,
'awe': 563,
'awesome': 564,
'awesomeconsnot': 565,
'awesomei': 566,
'awesomely': 567,
'awhile': 568,
'awry': 569,
'ax': 570,
'axiom': 571,
'axxholes': 572,
'ayyy': 573,
'az': 574,
'b': 575,
```

```
'babies': 576,
'baby': 577,
'babythe': 578,
'back': 579,
'backdecember': 580,
'backed': 581,
'backedup': 582,
'backfree': 583,
'background': 584,
'backgroundalso': 585,
'backing': 586,
'backorder': 587,
'backordered': 588,
'backs': 589,
'backside': 590,
'backstill': 591,
'backthis': 592,
'backtoback': 593,
'backup': 594,
'backups': 595,
'backward': 596,
'backwards': 597,
'bad': 598,
'bada': 599,
'badboy': 600,
'badboys': 601,
'badjust': 602,
'badly': 603,
'badupdate': 604,
'baffled': 605,
'bag': 606,
'balance': 607,
'ball': 608,
'ballpark': 609,
'bam': 610,
'bandwagon': 611,
'bandwidth': 612,
'bang': 613,
'bango': 614,
'banking': 615,
'banner': 616,
'bar': 617,
'bare': 618,
'barely': 619,
'bargain': 620,
'bargainbasement': 621,
'bargan': 622,
'barn': 623,
'base': 624,
'based': 625,
'basic': 626,
'basically': 627,
'basics': 628,
'basis': 629,
'basketball': 630,
'bass': 631,
'bast': 632,
'basta': 633,
'bat': 634,
'batch': 635,
'batches': 636,
'batchgopro': 637,
'batchive': 638,
'batteries': 639,
```

```
'battery': 640,
'bay': 641,
'bays': 642,
'bb': 643,
'bbboth': 644,
'bconvience': 645,
'bcuz': 646,
'beagle': 647,
'bean': 648,
'bear': 649,
'bearing': 650,
'beast': 651,
'beat': 652,
'beatifully': 653,
'beating': 654,
'beats': 655,
'beautiful': 656,
'beautifully': 657,
'beauty': 658,
'became': 659,
'becasue': 660,
'becauseive': 661,
'become': 662,
'becomes': 663,
'becoming': 664,
'becuase': 665,
'becuse': 666,
'bed': 667,
'beef': 668,
'beefy': 669,
'beforecon': 670,
'beforehand': 671,
'beforemicrosd': 672,
'beforeso': 673,
'beg': 674,
'began': 675,
'begin': 676,
'beginning': 677,
'begins': 678,
'begun': 679,
'behave': 680,
'behaved': 681,
'behaves': 682,
'behavior': 683,
'behavioral': 684,
'behind': 685,
'behold': 686,
'beijing': 687,
'beingit': 688,
'beit': 689,
'beleive': 690,
'believ': 691,
'believe': 692,
'believer': 693,
'beloved': 694,
'belowfirstly': 695,
'bemuch': 696,
'bench': 697,
'benchmark': 698,
'benchmarked': 699,
'benchmarking': 700,
'benchmarks': 701,
'benchmarksettings': 702,
'benchmarkspurchased': 703,
```

```
'bend': 704,
'bene': 705,
'benefit': 706,
'benefits': 707,
'benissimo': 708,
'beside': 709,
'besides': 710,
'best': 711,
'bestbuy': 712,
'besti': 713,
'bestthen': 714,
'bet': 715,
'better': 716,
'betteri': 717,
'betterthanks': 718,
'betterwith': 719,
'beware': 720,
'bewareupdate': 721,
'bext': 722,
'beyond': 723,
'beyondhave': 724,
'bg': 725,
'biased': 726,
'bible': 727,
'bid': 728,
'bienahora': 729,
'big': 730,
'bigbox': 731,
'bigger': 732,
'biggerolder': 733,
'biggest': 734,
'biggestcapacity': 735,
'biggie': 736,
'bike': 737,
'bill': 738,
'billing': 739,
'billion': 740,
'bin': 741,
'binary': 742,
'bindings': 743,
'bingo': 744,
'bins': 745,
'bionic': 746,
'bionici': 747,
'birthday': 748,
'bit': 749,
'bites': 750,
'bitlocker': 751,
'bitrate': 752,
'bitrates': 753,
'bits': 754,
'bitty': 755,
'bizillion': 756,
'bla': 757,
'black': 758,
'blackberry': 759,
'blackbox': 760,
'blackfriday': 761,
'blackno': 762,
'blackto': 763,
'blackvue': 764,
'blackwith': 765,
'blade': 766,
'blah': 767,
```

```
'blame': 768,
'blamed': 769,
'blank': 770,
'blankit': 771,
'blankunformatted': 772,
'blazaing': 773,
'blaze': 774,
'blazes': 775,
'blazing': 776,
'bleeding': 777,
'blend': 778,
'blew': 779,
'blink': 780,
'blinked': 781,
'blinks': 782,
'blip': 783,
'blipped': 784,
'blips': 785,
'bliss': 786,
'blissful': 787,
'blister': 788,
'bloated': 789,
'bloatware': 790,
'block': 791,
'blocked': 792,
'blocking': 793,
'blocks': 794,
'blocky': 795,
'blogs': 796,
'blow': 797,
'blowing': 798,
'blown': 799,
'blows': 800,
'blue': 801,
'blueand': 802,
'bluetooth': 803,
'bluetoothcompatible': 804,
'bluray': 805,
'blurays': 806,
'bn': 807,
'bnever': 808,
'board': 809,
'boards': 810,
'boat': 811,
'bobs': 812,
'body': 813,
'bogged': 814,
'bogs': 815,
'bogus': 816,
'bomb': 817,
'bone': 818,
'bonestock': 819,
'bonus': 820,
'bonusnow': 821,
'book': 822,
'booklet': 823,
'bookmarks': 824,
'books': 825,
'bookshelf': 826,
'boost': 827,
'boosted': 828,
'boosts': 829,
'boostwith': 830,
'boot': 831,
```

```
'bootable': 832,
'booted': 833,
'bootfor': 834,
'booting': 835,
'bootleg': 836,
'bootlegged': 837,
'bootloader': 838,
'boots': 839,
'bootso': 840,
'bootthis': 841,
'bootup': 842,
'bored': 843,
'boredom': 844,
'boring': 845,
'born': 846,
'borrowed': 847,
'bother': 848,
'bothered': 849,
'bothernow': 850,
'bothso': 851,
'boththis': 852,
'bottleneck': 853,
'bottlenecked': 854,
'bottlenecking': 855,
'bottlenecks': 856,
'bottom': 857,
'bough': 858,
'bought': 859,
'bougth': 860,
'bounce': 861,
'bounced': 862,
'bouncing': 863,
'boundary': 864,
'bout': 865,
'bouth': 866,
'bouthg': 867,
'boverall': 868,
'box': 869,
'boxand': 870,
'boxes': 871,
'boxno': 872,
'boxsky': 873,
'boxthe': 874,
'boy': 875,
'boyfriend': 876,
'brag': 877,
'bragged': 878,
'brain': 879,
'brainer': 880,
'brainernote': 881,
'brainerthe': 882,
'bran': 883,
'brand': 884,
'brandaaaaaaaa': 885,
'brandagain': 886,
'branded': 887,
'brandi': 888,
'branding': 889,
'brandname': 890,
'brands': 891,
'brandsi': 892,
'brandversion': 893,
'brandyou': 894,
'bravo': 895,
```

```
'bread': 896,
'break': 897,
'breakdown': 898,
'breaker': 899,
'breaking': 900,
'breaks': 901,
'breathe': 902,
'breathing': 903,
'breed': 904,
'breeze': 905,
'brew': 906,
'brick': 907,
'bricked': 908,
'bridge': 909,
'brief': 910,
'briefly': 911,
'bright': 912,
'brighter': 913,
'brilliant': 914,
'brilliantly': 915,
'bring': 916,
'brings': 917,
'brisk': 918,
'brittle': 919,
'broadcasts': 920,
'broke': 921,
'broken': 922,
'brokenalso': 923,
'brokencorruptunusable': 924,
'brotha': 925,
'brother': 926,
'brothers': 927,
'brought': 928,
'brown': 929,
'browse': 930,
'browser': 931,
'browsing': 932,
'bruce': 933,
'brutally': 934,
'bs': 935,
'bsi': 936,
'btw': 937,
'btwshame': 938,
'bublbe': 939,
'buck': 940,
'bucks': 941,
'budget': 942,
'budgetso': 943,
'buena': 944,
'bueno': 945,
'buff': 946,
'buffer': 947,
'buffering': 948,
'bufferinghiccupspauses': 949,
'buffers': 950,
'bufferthere': 951,
'bug': 952,
'bugger': 953,
'buggercons': 954,
'buggy': 955,
'bugs': 956,
'build': 957,
'building': 958,
'buildings': 959,
```

```
 'built': 960,
 'builtin': 961,
 'buit': 962,
 'bujgger': 963,
 'bulit': 964,
 'bulk': 965,
 'bullcrap': 966,
 'bulletproof': 967,
 'bum': 968,
 'bummed': 969,
 'bummer': 970,
 'bump': 971,
 'bumps': 972,
 'bunch': 973,
 'bunches': 974,
 'bundle': 975,
 'bundling': 976,
 'bunk': 977,
 'bur': 978,
 'burchase': 979,
 'burn': 980,
 'burned': 981,
 'burner': 982,
 'burning': 983,
 'burst': 984,
 'bus': 985,
 'business': 986,
 'bust': 987,
 'busted': 988,
 'busy': 989,
 'butsmall': 990,
 'butt': 991,
 'butter': 992,
 'butthis': 993,
 'button': 994,
 'butwhat': 995,
 'buy': 996,
 'buyer': 997,
 'buyers': 998,
 'buyi': 999,
 ...}
```

2.1c

Please implement a function named reduce_to_k_dim(M) performs dimensionality reduction on the matrix M to produce k-dimensional embeddings and returns the new matrix M_reduced. Use SVD (use the implementation of Truncated SVD in sklearn sklearn.decomposition.TruncatedSVD, set n_iters = 10) to take the top k components and produce a new matrix of k-dimensional embeddings.

In [13]:
```
# Import necessary library
from sklearn.decomposition import TruncatedSVD
```

In [14]:
```
def reduce_to_k_dim(M, k=2):

    """ Reduce a co-occurence count matrix of dimensionality (num_corpus_wor
        to a matrix of dimensionality (num_corpus_words, k) using the follow

        Params:
            M (numpy matrix of shape (number of unique words in the corpus ,
            k (int): embedding size of each word after dimension reduction
        Return:
```

```
            M_fit_svd (numpy matrix of shape (number of corpus words, k)): r
                  In terms of the SVD from math class, this actually returns
    """

    # Create a Singular Value Decomposition model with the value of n_iters
    svd = TruncatedSVD(n_components=k, n_iter=10, random_state=100)

    # Fit the svd model to the matrix M
    M_fit_svd = svd.fit_transform(M)

    return M_fit_svd

M_reduced_count_based = reduce_to_k_dim(M_count_based)
M_reduced_count_based
```

Out[14]:
```
array([[ 0.12846658,  0.10059098],
       [ 0.18762799,  0.03384236],
       [ 1.64781196, -1.15541128],
       ...,
       [ 1.07253206, -0.60830461],
       [ 1.27791587, -0.22853538],
       [ 0.09724572,  0.02260818]])
```

In [15]:
```
M_reduced_count_based.shape
```

Out[15]:
```
(9697, 2)
```

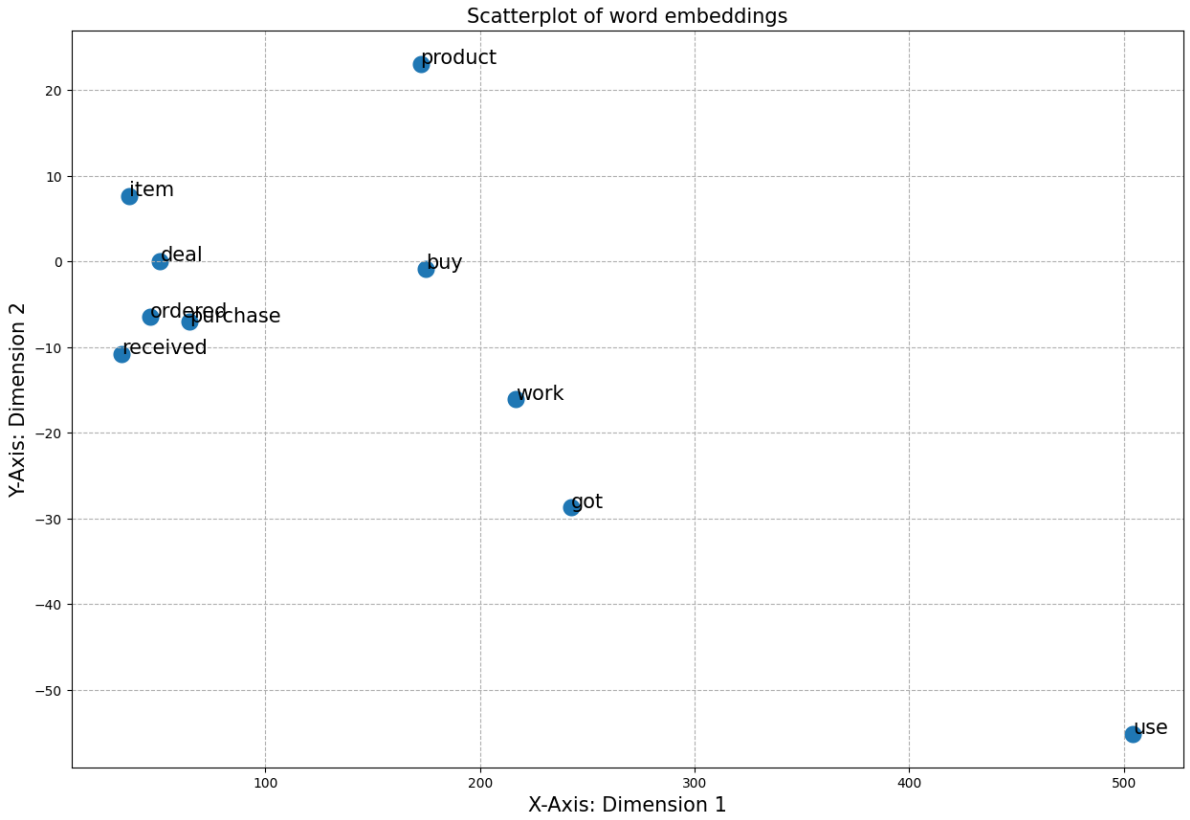
2.1d

Implement plot_embeddings(M_reduced, word2index, words_to_plot) to plot in a scatterplot the embeddings of the words specified in the list 'words_to_plot'. Here, 'M_reduced' is the matrix of 2-dimensional word embeddings obtained in question c. word2index is the dictionary that maps words to indices for the embedding matrix obtained in question b. Use the implemented function to get the plot for the following list of words words_to_plot=['purchase', 'buy', 'work', 'got', 'ordered', 'received', 'product', 'item', 'deal', 'use'], and show the plot.

In [16]:
```
# import necessary library for plotting
import matplotlib.pyplot as plt
import seaborn as sns
```

In [17]:
```
def plot_embeddings(M_reduced, word2index, words_to_plot):

    """ Plot in a scatterplot the embeddings of the words specified in the l
        NOTE: do not plot all the words listed in M_reduced / word2ind.
        Include a label next to each point.
        Params:
            M_reduced (numpy matrix of shape (number of unique words in the
            word2index (dict): dictionary that maps word to indices for matr
            words_to_plot (list of strings): words whose embeddings we want
    """

    # Get the index of each word in the list of words_to_plot from word2inde
    # Iterate through each word in the list of words_to_plot and map with th
    words_to_plot_indices = []
    for word in words_to_plot:
        words_to_plot_indices.append(word2index[word])

    # Get the embeddings of only the group of words in "words_to_plot"
    embeddings_to_plot = M_reduced[words_to_plot_indices]
```

```
    # plot a scatterplot
    plt.figure(figsize=(15,10))
    sns.scatterplot(x=embeddings_to_plot[:, 0], y=embeddings_to_plot[:, 1],
    plt.grid(visible=True, which='major', linestyle='--', linewidth=0.8)

    # Add labels, and title on the plot
    for index, word in enumerate(words_to_plot):
        plt.annotate(word, (embeddings_to_plot[index, 0], embeddings_to_plot
    plt.xlabel('X-Axis: Dimension 1', fontsize=15)
    plt.ylabel('Y-Axis: Dimension 2', fontsize=15)
    plt.title('Scatterplot of word embeddings', fontsize=15)


words_to_plot = ['purchase', 'buy', 'work', 'got', 'ordered', 'received', 'p
plot_embeddings(M_reduced_count_based, word2index_count_based, words_to_plot
```



2.2 Prediction-based word vectors from GloVe

2.2a

Please use the provided load_embedding_model() function to load the GloVe embeddings. Note: If this is your first time to run these cells, i.e. download the embedding model, it will take a couple minutes to run. If you've run these cells before, rerunning them will load the model without redownloading it, which will take about 1 to 2 minutes.

In [18]:
```
# Download the embedding model
def load_embedding_model():

    """ Load GloVe Vectors
        Return:
            wv_from_bin: All 400000 embeddings, each lengh 200
    """

    # Download the embedding model (Only the first time)
    import gensim.downloader as api
```

```
    wv_from_bin = api.load("glove-wiki-gigaword-200")

    print("Loaded vocab size %i" % len(list(wv_from_bin.index_to_key)))

    return wv_from_bin

wv_from_bin = load_embedding_model()
```

```
Loaded vocab size 400000
```

2.2b

Select the words in the vocabulary returned in 2.1a and get the corresponding GloVe vectors. You can adapt the provided function get_matrix_of_vectors(wv_from_bin, required_words) to select the Glove vectors and put them in a matrix M.

In [19]:

```
def get_matrix_of_vectors(wv_from_bin, required_words):

    """ Put the GloVe vectors into a matrix M.
        Param:
            wv_from_bin: KeyedVectors object; the 400000 GloVe vectors loade
            required_words: the list of strings that contains all required v
        Return:
            M: numpy matrix shape (num words, 200) containing the vectors
            word2ind: dictionary mapping each word to its row number in M
    """
    # Get the vectors of the words from GloVe
    import random
    words = list(wv_from_bin.index_to_key)
    print("Shuffling words ...")
    random.seed(225)
    random.shuffle(words)
    words = words[:]
    print("Putting %i words into word2index and matrix M..." % len(required_
    word2index = {}
    M = []
    curIndex = 0
    default_vector = np.zeros(200)

    # Get the vectors of the required words
    for word in required_words:
        if word in words:
            M.append(wv_from_bin.get_vector(word))
            word2index[word] = curIndex
            curIndex += 1
        else:
            # Replace missing words with a defined default vector
            M.append(default_vector)
            # Map the relationship between key: word, and value: index of th
            word2index[word] = curIndex
            curIndex += 1
    M = np.stack(M)
    print("Done.")
    return M, word2index

M_pred_based, word2index_pred_based = get_matrix_of_vectors(wv_from_bin, cor
M_pred_based
```

```
Shuffling words ...
Putting 9697 words into word2index and matrix M...
Done.
```

Out[19]:
```
array([[ 0.22586   ,  0.055649  ,  0.28367999, ..., -0.28356999,
         0.36353001, -0.87370002],
       [ 0.23273   , -0.048693  ,  0.76813   , ..., -0.55684   ,
        -0.44095999,  0.20299999],
       [-0.075412  ,  0.34158   , -0.66105002, ...,  0.14579   ,
        -0.57616001, -0.022035  ],
       ...,
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [-0.35547999,  0.22649001, -0.37088999, ..., -0.062321  ,
         0.19923   , -0.15657   ],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ]])
```

In [20]:
```
M_pred_based.shape
```

Out[20]:
```
(9697, 200)
```

In [21]:
```
word2index_pred_based
```

Out[21]:

```
{'aac': 0,
 'aas': 1,
 'aba': 2,
 'abdroid': 3,
 'abilities': 4,
 'ability': 5,
 'able': 6,
 'aboutgood': 7,
 'abouti': 8,
 'aboutherehere': 9,
 'aboutto': 10,
 'abovei': 11,
 'abroad': 12,
 'abruptly': 13,
 'absolute': 14,
 'absolutely': 15,
 'abt': 16,
 'abuse': 17,
 'abused': 18,
 'abysmal': 19,
 'accdientally': 20,
 'accept': 21,
 'acceptable': 22,
 'acceptably': 23,
 'accepted': 24,
 'accepting': 25,
 'accepts': 26,
 'access': 27,
 'accessed': 28,
 'accesses': 29,
 'accessible': 30,
 'accessing': 31,
 'accessories': 32,
 'accessory': 33,
 'accessoryalternative': 34,
 'accident': 35,
 'accidentally': 36,
 'accidently': 37,
 'acclimated': 38,
 'accolades': 39,
 'accommodate': 40,
 'accomplish': 41,
 'accord': 42,
 'according': 43,
 'accordingly': 44,
 'account': 45,
 'accros': 46,
 'accurate': 47,
 'accuratei': 48,
 'accuratethe': 49,
 'ace': 50,
 'acer': 51,
 'acess': 52,
 'acheive': 53,
 'achieve': 54,
 'achieved': 55,
 'achievedusing': 56,
 'achieves': 57,
 'acitve': 58,
 'acknowledge': 59,
 'acknowledged': 60,
 'acontourroam': 61,
 'acquire': 62,
 'acquired': 63,
```

```
'acronis': 64,
'acronyms': 65,
'across': 66,
'act': 67,
'acting': 68,
'action': 69,
'actioncam': 70,
'activating': 71,
'active': 72,
'activity': 73,
'acts': 74,
'actual': 75,
'actuality': 76,
'actually': 77,
'ad': 78,
'adapater': 79,
'adapt': 80,
'adapted': 81,
'adapter': 82,
'adapterbased': 83,
'adapterbingo': 84,
'adapterbqzhjs': 85,
'adapterexchanging': 86,
'adapterfunny': 87,
'adapteri': 88,
'adapterit': 89,
'adapterive': 90,
'adapterlifetime': 91,
'adapterlol': 92,
'adapterother': 93,
'adapterprecautions': 94,
'adapterpretty': 95,
'adapters': 96,
'adaptersamsung': 97,
'adaptersandisk': 98,
'adaptershipping': 99,
'adaptersome': 100,
'adaptersuper': 101,
'adapterthis': 102,
'adapterwhich': 103,
'adapterwith': 104,
'adapterwithout': 105,
'adapterworks': 106,
'adaptor': 107,
'adaptornow': 108,
'adaptors': 109,
'adaptorsthat': 110,
'adaquate': 111,
'adata': 112,
'adatas': 113,
'add': 114,
'added': 115,
'adding': 116,
'addingdeletingmoving': 117,
'addition': 118,
'additional': 119,
'additionals': 120,
'addon': 121,
'address': 122,
'addressed': 123,
'adds': 124,
'addsi': 125,
'adequate': 126,
'adequatei': 127,
```

```
'adequately': 128,
'adjust': 129,
'admiral': 130,
'admit': 131,
'admits': 132,
'admitted': 133,
'admittedly': 134,
'adn': 135,
'adopter': 136,
'adopting': 137,
'adpator': 138,
'ads': 139,
'adult': 140,
'advance': 141,
'advanced': 142,
'advancing': 143,
'advantage': 144,
'adventure': 145,
'adventures': 146,
'adventuresthis': 147,
'adversities': 148,
'advertise': 149,
'advertised': 150,
'advertisedgood': 151,
'advertisedi': 152,
'advertisedif': 153,
'advertisedthis': 154,
'advertisedwhat': 155,
'advertisement': 156,
'advertisements': 157,
'advertises': 158,
'advertising': 159,
'advertized': 160,
'advice': 161,
'advise': 162,
'advised': 163,
'adware': 164,
'aelago': 165,
'aerobic': 166,
'affect': 167,
'affected': 168,
'affecting': 169,
'afford': 170,
'affordable': 171,
'affpa': 172,
'afraid': 173,
'aft': 174,
'afternoon': 175,
'afterward': 176,
'afterwards': 177,
'againalso': 178,
'againas': 179,
'againdecember': 180,
'againedit': 181,
'againi': 182,
'againif': 183,
'againit': 184,
'againlolgo': 185,
'againspeed': 186,
'againwriting': 187,
'age': 188,
'agent': 189,
'agents': 190,
'aging': 191,
```

```
'ago': 192,
'agoi': 193,
'agothankfully': 194,
'agoworks': 195,
'agptek': 196,
'agree': 197,
'agreed': 198,
'ah': 199,
'ahaha': 200,
'ahead': 201,
'aiiiighsuperspeedy': 202,
'ainol': 203,
'aint': 204,
'air': 205,
'airdroid': 206,
'airplane': 207,
'airport': 208,
'aka': 209,
'akingston': 210,
'al': 211,
'alamcenar': 212,
'alaska': 213,
'alaskai': 214,
'albeit': 215,
'album': 216,
'albums': 217,
'aldo': 218,
'alec': 219,
'alert': 220,
'alerted': 221,
'alerting': 222,
'alerts': 223,
'alien': 224,
'alike': 225,
'alla': 226,
'allbased': 227,
'allblack': 228,
'alldaymalltm': 229,
'alleged': 230,
'allegedly': 231,
'allexcellent': 232,
'allgb': 233,
'allhave': 234,
'alli': 235,
'allinone': 236,
'allinstallation': 237,
'allocated': 238,
'allocation': 239,
'allot': 240,
'allotted': 241,
'allow': 242,
'allowed': 243,
'allowing': 244,
'allows': 245,
'allpurpose': 246,
'allready': 247,
'allsome': 248,
'allthough': 249,
'almost': 250,
'almsot': 251,
'alone': 252,
'alonethe': 253,
'along': 254,
'alot': 255,
```

```
'already': 256,
'alreadythis': 257,
'alright': 258,
'also': 259,
'alsobuying': 260,
'alt': 261,
'alta': 262,
'altered': 263,
'alternately': 264,
'alternative': 265,
'although': 266,
'altogether': 267,
'altogetherwhen': 268,
'altough': 269,
'aluminum': 270,
'alway': 271,
'always': 272,
'alwayspros': 273,
'alzheimers': 274,
'amamzon': 275,
'amazed': 276,
'amazementit': 277,
'amazes': 278,
'amazing': 279,
'amazingi': 280,
'amazingly': 281,
'amazingsandisk': 282,
'amazon': 283,
'amazonas': 284,
'amazoncom': 285,
'amazonconits': 286,
'amazonenjoy': 287,
'amazonfrustrating': 288,
'amazoni': 289,
'amazonpackage': 290,
'amazonprime': 291,
'amazons': 292,
'amazonseeing': 293,
'amazonsincerelykj': 294,
'amazonso': 295,
'amazonthe': 296,
'amazonthey': 297,
'america': 298,
'amis': 299,
'ammount': 300,
'amon': 301,
'among': 302,
'amonths': 303,
'amount': 304,
'amounts': 305,
'ample': 306,
'ampliamente': 307,
'ana': 308,
'anbd': 309,
'andi': 310,
'andiod': 311,
'andloaded': 312,
'andoid': 313,
'andor': 314,
'andriod': 315,
'androd': 316,
'android': 317,
'androidread': 318,
'androidspecific': 319,
```

```
'ands': 320,
'andthats': 321,
'andy': 322,
'anecdotal': 323,
'anelago': 324,
'angle': 325,
'angryto': 326,
'anime': 327,
'anker': 328,
'anniversary': 329,
'announced': 330,
'announcement': 331,
'annoyance': 332,
'annoyed': 333,
'annoying': 334,
'anomaly': 335,
'another': 336,
'anotherbigger': 337,
'anothergreat': 338,
'answer': 339,
'answered': 340,
'answers': 341,
'answersthanks': 342,
'anteriormente': 343,
'anticipated': 344,
'anticipating': 345,
'anticipation': 346,
'antishoplifting': 347,
'antivirus': 348,
'antutu': 349,
'anyall': 350,
'anybody': 351,
'anycorrupted': 352,
'anydvd': 353,
'anyhow': 354,
'anyjust': 355,
'anymore': 356,
'anymorehope': 357,
'anymoreif': 358,
'anymoreone': 359,
'anyne': 360,
'anynot': 361,
'anyone': 362,
'anyoneps': 363,
'anyproblems': 364,
'anythin': 365,
'anything': 366,
'anythingvery': 367,
'anythingworks': 368,
'anytime': 369,
'anyway': 370,
'anywayany': 371,
'anyways': 372,
'anywhere': 373,
'anywho': 374,
'aokp': 375,
'aosp': 376,
'apapter': 377,
'apart': 378,
'apartment': 379,
'apex': 380,
'aplomb': 381,
'app': 382,
'apparent': 383,
```

```
'apparently': 384,
'appbrowsernetflixmusic': 385,
'appealed': 386,
'appealing': 387,
'appear': 388,
'appearance': 389,
'appeared': 390,
'appearing': 391,
'appears': 392,
'appendage': 393,
'apple': 394,
'appliance': 395,
'applicable': 396,
'application': 397,
'applicationas': 398,
'applicationi': 399,
'applications': 400,
'applied': 401,
'apply': 402,
'apposed': 403,
'appreadergb': 404,
'appreciate': 405,
'appreciated': 406,
'appreciates': 407,
'appreciative': 408,
'apprehensive': 409,
'approaching': 410,
'appropriate': 411,
'approval': 412,
'approve': 413,
'approved': 414,
'approx': 415,
'approximately': 416,
'apps': 417,
'appsd': 418,
'appsgames': 419,
'appsi': 420,
'apr': 421,
'april': 422,
'ar': 423,
'arc': 424,
'architecture': 425,
'archive': 426,
'archos': 427,
'arduous': 428,
'area': 429,
'areanyways': 430,
'areas': 431,
'arehaving': 432,
'arei': 433,
'arent': 434,
'argue': 435,
'arguing': 436,
'arise': 437,
'ariseit': 438,
'arises': 439,
'arm': 440,
'armed': 441,
'arose': 442,
'around': 443,
'aroundand': 444,
'aroundsc': 445,
'arounf': 446,
'arranca': 447,
```

```
'arrival': 448,
'arrivalwhat': 449,
'arrive': 450,
'arrived': 451,
'arrivedopting': 452,
'arrives': 453,
'art': 454,
'arthritis': 455,
'artiaga': 456,
'articles': 457,
'articleso': 458,
'asamsung': 459,
'ashtray': 460,
'asian': 461,
'aside': 462,
'asis': 463,
'asix': 464,
'ask': 465,
'asked': 466,
'aski': 467,
'asking': 468,
'asks': 469,
'asleep': 470,
'asneeded': 471,
'aspect': 472,
'asphalt': 473,
'aspirecarry': 474,
'ass': 475,
'assembly': 476,
'assess': 477,
'assessment': 478,
'asset': 479,
'assign': 480,
'assist': 481,
'association': 482,
'assorted': 483,
'assortment': 484,
'assume': 485,
'assumed': 486,
'assumedprosincredible': 487,
'assuming': 488,
'assurance': 489,
'assure': 490,
'assured': 491,
'assuringly': 492,
'astak': 493,
'astonishingly': 494,
'asus': 495,
'asusprime': 496,
'ati': 497,
'ativ': 498,
'ativpc': 499,
'atlanta': 500,
'atlantic': 501,
'atleast': 502,
'atop': 503,
'atore': 504,
'atrecover': 505,
'atrix': 506,
'atrocious': 507,
'att': 508,
'attached': 509,
'attaches': 510,
'attaching': 511,
```

```
'attachment': 512,
'attain': 513,
'attempt': 514,
'attempted': 515,
'attempting': 516,
'attempts': 517,
'attention': 518,
'attest': 519,
'atto': 520,
'attracted': 521,
'attractive': 522,
'attribute': 523,
'auction': 524,
'audible': 525,
'audio': 526,
'audiobooks': 527,
'audiovideo': 528,
'aug': 529,
'augmentation': 530,
'august': 531,
'authentic': 532,
'authenticate': 533,
'authenticity': 534,
'authenticthe': 535,
'authorized': 536,
'auththentic': 537,
'auto': 538,
'automatic': 539,
'automatically': 540,
'automotive': 541,
'automount': 542,
'autophoto': 543,
'autoscanned': 544,
'av': 545,
'avail': 546,
'available': 547,
'availablethe': 548,
'average': 549,
'averaged': 550,
'averages': 551,
'avg': 552,
'avoid': 553,
'avoided': 554,
'avoiding': 555,
'await': 556,
'awaiting': 557,
'aware': 558,
'awareif': 559,
'away': 560,
'awaygb': 561,
'awayroot': 562,
'awe': 563,
'awesome': 564,
'awesomeconsnot': 565,
'awesomei': 566,
'awesomely': 567,
'awhile': 568,
'awry': 569,
'ax': 570,
'axiom': 571,
'axxholes': 572,
'ayyy': 573,
'az': 574,
'b': 575,
```

```
'babies': 576,
'baby': 577,
'babythe': 578,
'back': 579,
'backdecember': 580,
'backed': 581,
'backedup': 582,
'backfree': 583,
'background': 584,
'backgroundalso': 585,
'backing': 586,
'backorder': 587,
'backordered': 588,
'backs': 589,
'backside': 590,
'backstill': 591,
'backthis': 592,
'backtoback': 593,
'backup': 594,
'backups': 595,
'backward': 596,
'backwards': 597,
'bad': 598,
'bada': 599,
'badboy': 600,
'badboys': 601,
'badjust': 602,
'badly': 603,
'badupdate': 604,
'baffled': 605,
'bag': 606,
'balance': 607,
'ball': 608,
'ballpark': 609,
'bam': 610,
'bandwagon': 611,
'bandwidth': 612,
'bang': 613,
'bango': 614,
'banking': 615,
'banner': 616,
'bar': 617,
'bare': 618,
'barely': 619,
'bargain': 620,
'bargainbasement': 621,
'bargan': 622,
'barn': 623,
'base': 624,
'based': 625,
'basic': 626,
'basically': 627,
'basics': 628,
'basis': 629,
'basketball': 630,
'bass': 631,
'bast': 632,
'basta': 633,
'bat': 634,
'batch': 635,
'batches': 636,
'batchgopro': 637,
'batchive': 638,
'batteries': 639,
```

```
'battery': 640,
'bay': 641,
'bays': 642,
'bb': 643,
'bbboth': 644,
'bconvience': 645,
'bcuz': 646,
'beagle': 647,
'bean': 648,
'bear': 649,
'bearing': 650,
'beast': 651,
'beat': 652,
'beatifully': 653,
'beating': 654,
'beats': 655,
'beautiful': 656,
'beautifully': 657,
'beauty': 658,
'became': 659,
'becasue': 660,
'becauseive': 661,
'become': 662,
'becomes': 663,
'becoming': 664,
'becuase': 665,
'becuse': 666,
'bed': 667,
'beef': 668,
'beefy': 669,
'beforecon': 670,
'beforehand': 671,
'beforemicrosd': 672,
'beforeso': 673,
'beg': 674,
'began': 675,
'begin': 676,
'beginning': 677,
'begins': 678,
'begun': 679,
'behave': 680,
'behaved': 681,
'behaves': 682,
'behavior': 683,
'behavioral': 684,
'behind': 685,
'behold': 686,
'beijing': 687,
'beingit': 688,
'beit': 689,
'beleive': 690,
'believ': 691,
'believe': 692,
'believer': 693,
'beloved': 694,
'belowfirstly': 695,
'bemuch': 696,
'bench': 697,
'benchmark': 698,
'benchmarked': 699,
'benchmarking': 700,
'benchmarks': 701,
'benchmarksettings': 702,
'benchmarkspurchased': 703,
```

```
'bend': 704,
'bene': 705,
'benefit': 706,
'benefits': 707,
'benissimo': 708,
'beside': 709,
'besides': 710,
'best': 711,
'bestbuy': 712,
'besti': 713,
'bestthen': 714,
'bet': 715,
'better': 716,
'betteri': 717,
'betterthanks': 718,
'betterwith': 719,
'beware': 720,
'bewareupdate': 721,
'bext': 722,
'beyond': 723,
'beyondhave': 724,
'bg': 725,
'biased': 726,
'bible': 727,
'bid': 728,
'bienahora': 729,
'big': 730,
'bigbox': 731,
'bigger': 732,
'biggerolder': 733,
'biggest': 734,
'biggestcapacity': 735,
'biggie': 736,
'bike': 737,
'bill': 738,
'billing': 739,
'billion': 740,
'bin': 741,
'binary': 742,
'bindings': 743,
'bingo': 744,
'bins': 745,
'bionic': 746,
'bionici': 747,
'birthday': 748,
'bit': 749,
'bites': 750,
'bitlocker': 751,
'bitrate': 752,
'bitrates': 753,
'bits': 754,
'bitty': 755,
'bizillion': 756,
'bla': 757,
'black': 758,
'blackberry': 759,
'blackbox': 760,
'blackfriday': 761,
'blackno': 762,
'blackto': 763,
'blackvue': 764,
'blackwith': 765,
'blade': 766,
'blah': 767,
```

```
'blame': 768,
'blamed': 769,
'blank': 770,
'blankit': 771,
'blankunformatted': 772,
'blazaing': 773,
'blaze': 774,
'blazes': 775,
'blazing': 776,
'bleeding': 777,
'blend': 778,
'blew': 779,
'blink': 780,
'blinked': 781,
'blinks': 782,
'blip': 783,
'blipped': 784,
'blips': 785,
'bliss': 786,
'blissful': 787,
'blister': 788,
'bloated': 789,
'bloatware': 790,
'block': 791,
'blocked': 792,
'blocking': 793,
'blocks': 794,
'blocky': 795,
'blogs': 796,
'blow': 797,
'blowing': 798,
'blown': 799,
'blows': 800,
'blue': 801,
'blueand': 802,
'bluetooth': 803,
'bluetoothcompatible': 804,
'bluray': 805,
'blurays': 806,
'bn': 807,
'bnever': 808,
'board': 809,
'boards': 810,
'boat': 811,
'bobs': 812,
'body': 813,
'bogged': 814,
'bogs': 815,
'bogus': 816,
'bomb': 817,
'bone': 818,
'bonestock': 819,
'bonus': 820,
'bonusnow': 821,
'book': 822,
'booklet': 823,
'bookmarks': 824,
'books': 825,
'bookshelf': 826,
'boost': 827,
'boosted': 828,
'boosts': 829,
'boostwith': 830,
'boot': 831,
```

```
'bootable': 832,
'booted': 833,
'bootfor': 834,
'booting': 835,
'bootleg': 836,
'bootlegged': 837,
'bootloader': 838,
'boots': 839,
'bootso': 840,
'bootthis': 841,
'bootup': 842,
'bored': 843,
'boredom': 844,
'boring': 845,
'born': 846,
'borrowed': 847,
'bother': 848,
'bothered': 849,
'bothernow': 850,
'bothso': 851,
'boththis': 852,
'bottleneck': 853,
'bottlenecked': 854,
'bottlenecking': 855,
'bottlenecks': 856,
'bottom': 857,
'bough': 858,
'bought': 859,
'bougth': 860,
'bounce': 861,
'bounced': 862,
'bouncing': 863,
'boundary': 864,
'bout': 865,
'bouth': 866,
'bouthg': 867,
'boverall': 868,
'box': 869,
'boxand': 870,
'boxes': 871,
'boxno': 872,
'boxsky': 873,
'boxthe': 874,
'boy': 875,
'boyfriend': 876,
'brag': 877,
'bragged': 878,
'brain': 879,
'brainer': 880,
'brainernote': 881,
'brainerthe': 882,
'bran': 883,
'brand': 884,
'brandaaaaaaaa': 885,
'brandagain': 886,
'branded': 887,
'brandi': 888,
'branding': 889,
'brandname': 890,
'brands': 891,
'brandsi': 892,
'brandversion': 893,
'brandyou': 894,
'bravo': 895,
```

```
'bread': 896,
'break': 897,
'breakdown': 898,
'breaker': 899,
'breaking': 900,
'breaks': 901,
'breathe': 902,
'breathing': 903,
'breed': 904,
'breeze': 905,
'brew': 906,
'brick': 907,
'bricked': 908,
'bridge': 909,
'brief': 910,
'briefly': 911,
'bright': 912,
'brighter': 913,
'brilliant': 914,
'brilliantly': 915,
'bring': 916,
'brings': 917,
'brisk': 918,
'brittle': 919,
'broadcasts': 920,
'broke': 921,
'broken': 922,
'brokenalso': 923,
'brokencorruptunusable': 924,
'brotha': 925,
'brother': 926,
'brothers': 927,
'brought': 928,
'brown': 929,
'browse': 930,
'browser': 931,
'browsing': 932,
'bruce': 933,
'brutally': 934,
'bs': 935,
'bsi': 936,
'btw': 937,
'btwshame': 938,
'bublbe': 939,
'buck': 940,
'bucks': 941,
'budget': 942,
'budgetso': 943,
'buena': 944,
'bueno': 945,
'buff': 946,
'buffer': 947,
'buffering': 948,
'bufferinghiccupspauses': 949,
'buffers': 950,
'bufferthere': 951,
'bug': 952,
'bugger': 953,
'buggercons': 954,
'buggy': 955,
'bugs': 956,
'build': 957,
'building': 958,
'buildings': 959,
```

```
 'built': 960,
 'builtin': 961,
 'buit': 962,
 'bujgger': 963,
 'bulit': 964,
 'bulk': 965,
 'bullcrap': 966,
 'bulletproof': 967,
 'bum': 968,
 'bummed': 969,
 'bummer': 970,
 'bump': 971,
 'bumps': 972,
 'bunch': 973,
 'bunches': 974,
 'bundle': 975,
 'bundling': 976,
 'bunk': 977,
 'bur': 978,
 'burchase': 979,
 'burn': 980,
 'burned': 981,
 'burner': 982,
 'burning': 983,
 'burst': 984,
 'bus': 985,
 'business': 986,
 'bust': 987,
 'busted': 988,
 'busy': 989,
 'butsmall': 990,
 'butt': 991,
 'butter': 992,
 'butthis': 993,
 'button': 994,
 'butwhat': 995,
 'buy': 996,
 'buyer': 997,
 'buyers': 998,
 'buyi': 999,
 ...}
```

2.2c

Use the function reduce_to_k_dim() you implemented in 1)c to reduce the vectors to 2 dimension. Similar to what you did in 1)c.

In [22]:
```
M_reduced_pred_based = reduce_to_k_dim(M_pred_based)
M_reduced_pred_based
```

Out[22]:
```
array([[-0.6654562 ,  1.90102906],
       [-1.06744851,  0.76112002],
       [ 0.02208927, -0.53159346],
       ...,
       [ 0.        ,  0.        ],
       [-0.16928242,  2.9489755 ],
       [ 0.        ,  0.        ]])
```

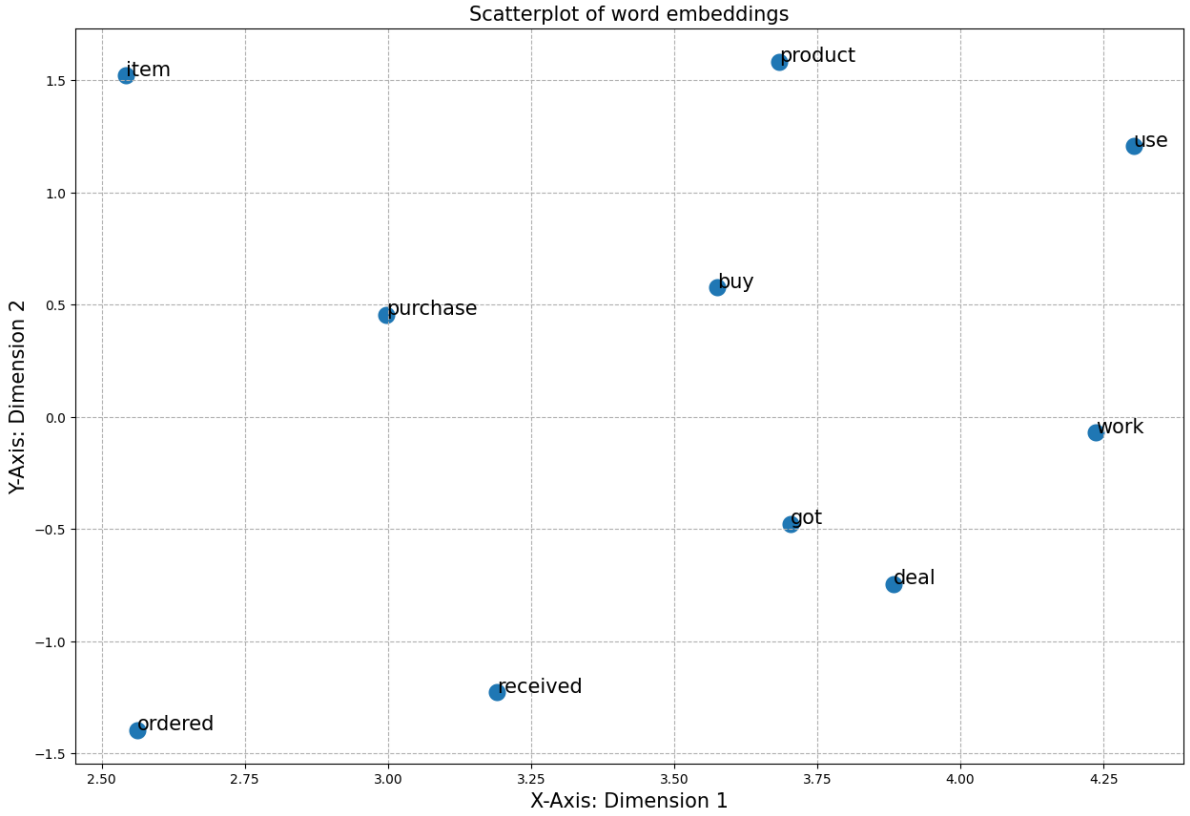
In [23]:
```
M_reduced_pred_based.shape
```

Out[23]:
```
(9697, 2)
```

2.2d

Use the plot_embeddings function in 1)d to get the plot for the same set of words in 1)d. Compare the differences of the plot in 1)d and 2)d, provide some analysis, and describe your findings.

In [24]:
```
plot_embeddings(M_reduced_pred_based, word2index_pred_based, words_to_plot)
```



Compare the difference of the scatter plot between Count-Based and Prediction-Based

In [25]:
```
# Get some information for summary

# The number of each plotted word
num_plot_words = {}
for word in words_to_plot:
    num_plot_words[word] = 0

for review in dataset['reviewText']:
    for word in review:
        if word in num_plot_words:
            num_plot_words[word] = num_plot_words.get(word, 0) + 1

# The total number of words in the entire reviews
total_words = 0
for review in dataset['reviewText']:
    total_words += len(review)

print(num_plot_words)
print("\nTotal words of the entire reviews:", total_words)
```

```
{'purchase': 166, 'buy': 455, 'work': 485, 'got': 568, 'ordered': 104, 'rec
eived': 78, 'product': 529, 'item': 119, 'deal': 129, 'use': 1115}

Total words of the entire reviews: 124713
```

From the scatter plots above,
1) Count-based word vectors with co-occurrence matrix

- Words tend to cluster together.
- Words that share similar frequencies of occurrence often form clusters, as demonstrated by the words such as "item," "deal," "purchase," "ordered," and "received."

2) Prediction-based word vectors from GloVe

- Words do not tend to cluster together.

Therefore, count-based word vectors with co-occurrence matrix emphasizes word frequency, leading to the observation that words with similar frequency tend to cluster together as shown in the plot. In contrast, prediction-based word vectors from GloVe focuses on the meanings of words, however the plot does not display clear clusters which may be caused by limited context or data and low frequencies of occurrence of words.

Task 2: Sentiment Classification Algorithms

3. Perform sentiment analysis with classification

3.1 Review embeddings

Similar to what you did in 2.2c, use the function reduce_to_k_dim() you implemented in 2.1c to reduce the vectors to 128 dimension. Based on the word embeddings, get the review embedding by taking the average of the word embeddings in each review. Write a function for getting review embeddings for each review.

In [26]:
```
def get_average_embeddings(M_reduced, dataset, word2index):

    """ A function to get review embeddings for each review
        by taking the average of the word embedding in each review
        Params:
            M_reduced: matrix of 128-dimentional word embeddings
            dataset: the original review data that has already tokenized the
            word2index: a dictionary that maps words to indices
        Return:
            review_embeddings: the review embedding from taking the average
    """

    # Create a list to store the review embeddings
    review_embeddings = np.zeros((len(dataset), len(M_reduced[0])))

    # Iterate through the list of tokenized text
    for index, review in enumerate(dataset['reviewText']):

        for j in range(len(M_reduced[0])):
            # Define variables to calculate an average value
            sum_embeddings = 0
            count = 0

            # Iterate through the list of each review
```

```
            for word in review:
                word_index = word2index[word]
                word_vector = M_reduced[word_index][j]
                sum_embeddings += word_vector
                count += 1

            # Get the average of the word embeddings
            average_embeddings = sum_embeddings / count
            review_embeddings[index][j] = average_embeddings

    return review_embeddings


# Reduce the dimentionality of matrix M with k = 128
M_reduced_pred_based_k128 = reduce_to_k_dim(M_pred_based, k=128)

# Call a function "get_average_embeddings" to get the average of words embe
review_embeddings = get_average_embeddings(M_reduced_pred_based_k128, datase
review_embeddings_dataframe = pd.DataFrame(review_embeddings)
new_dataset = pd.concat([dataset, review_embeddings_dataframe], axis=1)
new_dataset
```

Out[26]:

	overall	reviewText	0	1	2	3	4	5
0	4	[issues]	3.921470	-0.102500	-0.527179	-1.721730	-0.403041	0.204319
1	5	[purchased, device, worked, advertised, never,...	2.915476	0.734701	0.114749	0.642308	-0.556564	-0.429263
2	4	[works, expected, sprung, higher, capacity, th...	2.891930	0.262045	0.134002	0.318028	0.173038	0.128958
3	5	[think, worked, greathad, diff, bran, gb, card...	2.482578	-0.074888	-0.165305	0.262743	-0.277060	-0.174163
4	5	[bought, retail, packaging, arrived, legit, or...	2.396886	0.247296	-0.286311	0.697848	-0.362116	-0.350881
...	...	...	...	...	...	...	...	...
4910	1	[bought, sandisk, gb, class, use, htc, inspire...	2.299032	0.451168	0.156860	0.409932	-0.530156	-0.509213
4911	5	[used, extending, capabilities, samsung, galax...	2.625930	0.550545	-0.574834	0.070337	-0.016765	-0.296078
4912	5	[great, card, fast, reliable, comes, optional,...	2.745499	0.855996	-0.254797	0.447117	-0.383562	-0.146923
4913	5	[good, amount, space, stuff, want, fits, gopro...	2.974746	0.635473	0.868006	0.256999	-0.209651	-0.036941
4914	5	[ive, heard, bad, things, gb, micro, sd, card,...	2.385932	0.498374	-0.024667	0.297257	-0.032504	-0.253079

4915 rows × 130 columns

3.2 Models and 3.3 Evaluation

I will implement Logistic Regression, with L2 regularization and A Neural Network (NN) model for sentiment analysis. Then, I will evaluate the performance in predictions of these 2 models, using sklearn, Tensorflow, and Keras packages.

In this case, I will perform sentiment analysis with classification and I will simplify the task as a binary classification problem. I will consider positive reviews (Score 4 and 5) as 1, and negative reviews (Score 1, 2, and 3) as 0.

In [27]:
```
# Convert the target variable from multi-class to binary-class
new_dataset['overall'] [new_dataset['overall'] <= 3] = 0
new_dataset['overall'] [new_dataset['overall'] > 3] = 1
new_dataset
```

```
<ipython-input-27-892b68022222>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  new_dataset['overall'] [new_dataset['overall'] <= 3] = 0
<ipython-input-27-892b68022222>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  new_dataset['overall'] [new_dataset['overall'] > 3] = 1
```

Out[27]:

	overall	reviewText	0	1	2	3	4	5
0	1	[issues]	3.921470	-0.102500	-0.527179	-1.721730	-0.403041	0.204319
1	1	[purchased, device, worked, advertised, never,...	2.915476	0.734701	0.114749	0.642308	-0.556564	-0.429263
2	1	[works, expected, sprung, higher, capacity, th...	2.891930	0.262045	0.134002	0.318028	0.173038	0.128958
3	1	[think, worked, greathad, diff, bran, gb, card...	2.482578	-0.074888	-0.165305	0.262743	-0.277060	-0.174163
4	1	[bought, retail, packaging, arrived, legit, or...	2.396886	0.247296	-0.286311	0.697848	-0.362116	-0.350881
...	...	...	...	...	...	...	...	...
4910	0	[bought, sandisk, gb, class, use, htc, inspire...	2.299032	0.451168	0.156860	0.409932	-0.530156	-0.509213
4911	1	[used, extending, capabilities, samsung, galax...	2.625930	0.550545	-0.574834	0.070337	-0.016765	-0.296078
4912	1	[great, card, fast, reliable, comes, optional,...	2.745499	0.855996	-0.254797	0.447117	-0.383562	-0.146923
4913	1	[good, amount, space, stuff, want, fits, gopro...	2.974746	0.635473	0.868006	0.256999	-0.209651	-0.036941
4914	1	[ive, heard, bad, things, gb, micro, sd, card,...	2.385932	0.498374	-0.024667	0.297257	-0.032504	-0.253079

4915 rows × 130 columns

In [28]:

```
# Import necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accurac
```

Logistic Regression, with L2 Regularization

In [29]:

```
def imp_log_reg(X_train, y_train, X_val, y_val, X_test, y_test):

    """ A function to implement logistic regression
        Params:
            X_train: input features of the dataset for training data
            y_train: a target variable of the dataset for training data
            X_val: input features for validating data
            y_val: a target variable for validating data
            X_test: input features for testing data
            y_test: a target variable of the dataset for testing data
        Return:
            accuracy: the prediction accuracy of the model
            auc_result: the auc score of the model
            classification_report: a classification report displays the perf
            confusion_matrix: a confusion matrix displays the performance of
    """
    # Define the list of learning rate value to validate the model
    best_accuracy = 0
    best_c = None
    c_params = [0.001, 0.01, 0.1, 1, 10, 100]

    # Iterate through the list of learning rate parameters to find the best
    for c in c_params:
        model = LogisticRegression(multi_class='auto', penalty='l2', C=c)
        model.fit(X_train, y_train)

        # Validate the model with validate data to find the best learning ra
        y_val_pred = model.predict(X_val)
        val_accuracy = accuracy_score(y_val, y_val_pred)
        print("Validation Accuracy for C =", c, ":", val_accuracy)

        if val_accuracy > best_accuracy:
            best_accuracy = val_accuracy
            best_c = c

    print("The best value of parameter C is", best_c)
    # Create a logistic regression model with L2 regularization
    model = LogisticRegression(penalty='l2', C=best_c, max_iter=40)

    # Train the model on the training datas
    model.fit(X_train, y_train)

    # Test the model to predict the desired variable
    y_pred = model.predict(X_test)

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)
    auc_result = roc_auc_score(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    return accuracy, auc_result, class_report, conf_matrix

# Define features and target variables
X = new_dataset.drop(columns=['overall', 'reviewText'])
y = pd.DataFrame(new_dataset['overall'])

# Split the dataset into 3 parts including, train data, validation data, and
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, ran
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.
```

```python
accuracy, auc_result, class_report, conf_matrix = imp_log_reg(X_train, y_tra
print("accuracy:", accuracy)
print("auc score", auc_result)
print("\nclassification report:")
print(class_report)
print("\nconfusion matrix:")
print(conf_matrix)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: D
ataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using rave
l().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: D
ataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using rave
l().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: D
ataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using rave
l().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: D
ataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using rave
l().
  y = column_or_1d(y, warn=True)
Validation Accuracy for C = 0.001 : 0.9038461538461539
Validation Accuracy for C = 0.01 : 0.9038461538461539
Validation Accuracy for C = 0.1 : 0.9072398190045249
Validation Accuracy for C = 1 : 0.919683257918552
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: D
ataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using rave
l().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:4
58: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: D
ataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using rave
l().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:4
58: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: D
ataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using rave
l().
  y = column_or_1d(y, warn=True)
Validation Accuracy for C = 10 : 0.9321266968325792
Validation Accuracy for C = 100 : 0.9264705882352942
The best value of parameter C is 10
accuracy: 0.9292929292929293
auc score 0.7215909090909092

classification report:
              precision    recall  f1-score   support

           0       0.83      0.45      0.59        11
           1       0.94      0.99      0.96        88

    accuracy                           0.93        99
   macro avg       0.88      0.72      0.77        99
weighted avg       0.92      0.93      0.92        99


confusion matrix:
[[ 5  6]
 [ 1 87]]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:4
58: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession
  n_iter_i = _check_optimize_result(
```

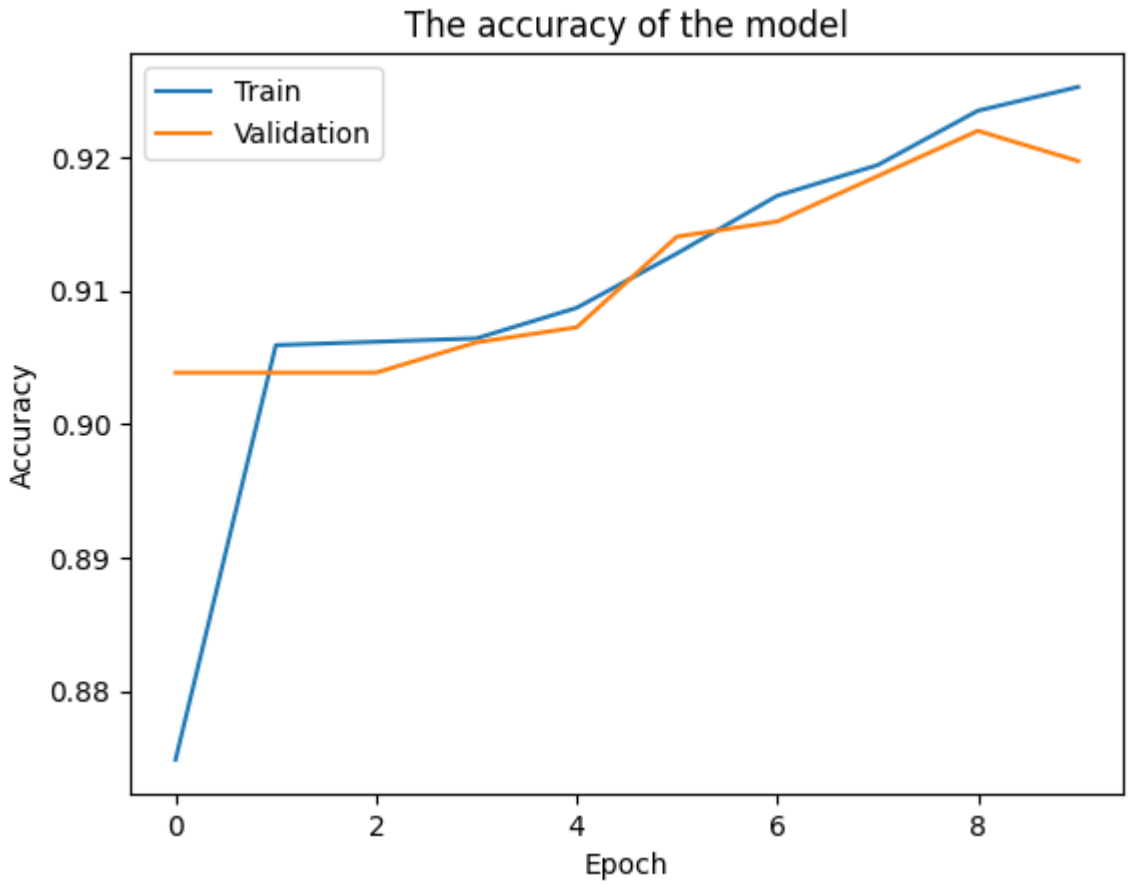
A Neural Network (NN) model for sentiment classification

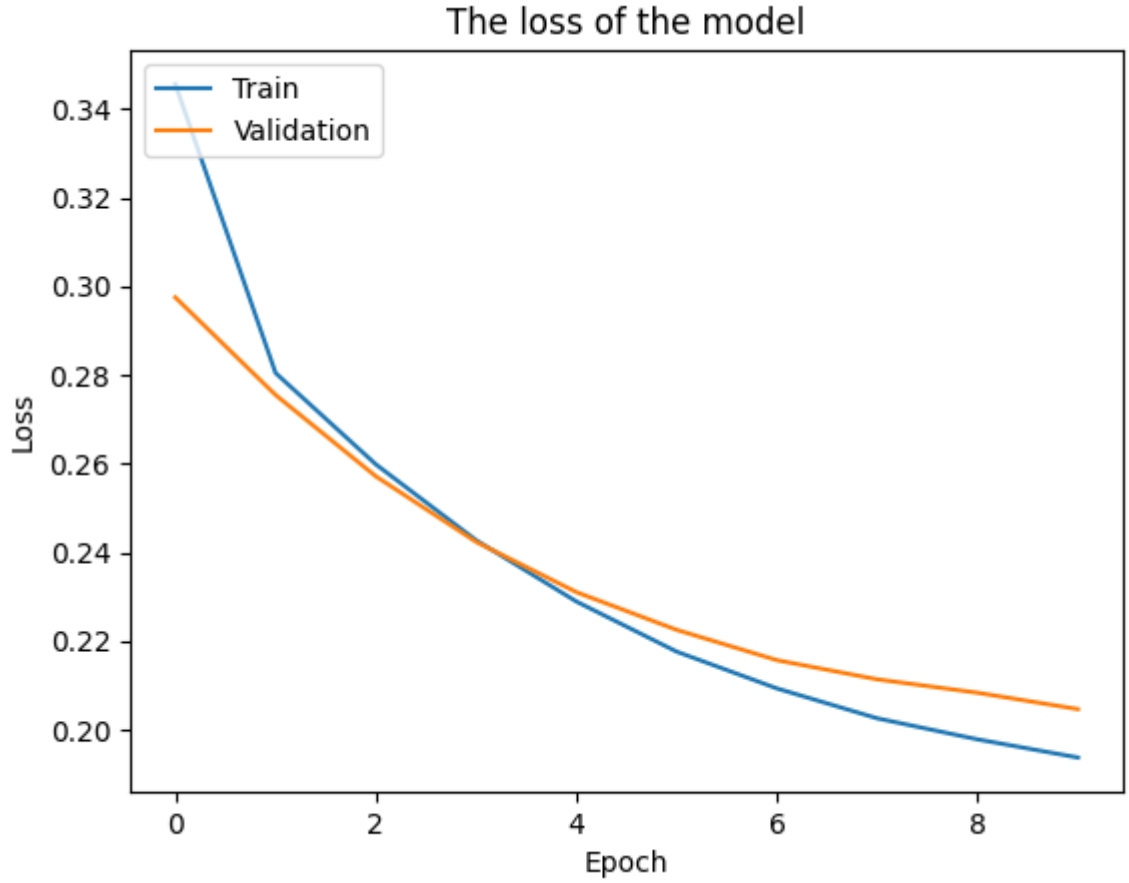
In [30]:
```python
# Import necessary libraries
import tensorflow as tf
from tensorflow import keras
```

In [31]:
```python
def imp_nn(X_train, y_train, X_val, y_val, X_test, y_test):
    """ A function to build and implement a neural network for sentiment ana
        Params:
            X_train: input features of the dataset for training data
            y_train: a target variable of the dataset for training data
            X_val: input features for validating data
            y_val: a target variable for validating data
            X_test: input features for testing data
            y_test: a target variable of the dataset for testing data
        Return:
            accuracy: the prediction accuracy of the model
            auc_result: the auc score of the model
            classification_report: a classification report displays the perf
            confusion_matrix: a confusion matrix displays the performance of
    """

    # Defines a neural network model
    model = keras.Sequential([
        keras.layers.Input(shape=(128,)), # Set input layer
        keras.layers.Dense(64, activation='sigmoid'), # Set hidden layer
        keras.layers.Dense(2, activation='sigmoid') # Set output layer
    ])

    # Compile the model before training the model to train dataset.
    model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",

    # Train the model
    history = model.fit(X_train, y_train, epochs=10, validation_data=(X_val,

    # Print the loss and accuracy after testing the model.
    loss, accuracy = model.evaluate(X_test, y_test)
    print(f"Testing loss: {loss}")
    print(f"Testing accuracy: {accuracy}")

    # Plot the graph to see the training and validation accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('The accuracy of the model')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.show()

    # Plot the graph to see the training and validation loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
```

```
    plt.title('The loss of the model')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.show()

    # Test the model to predict the desired variable
    y_pred_probs = model.predict(X_test)
    y_pred = np.argmax(y_pred_probs, axis=1)

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)
    auc_result = roc_auc_score(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    return accuracy, auc_result, class_report, conf_matrix

accuracy, auc_result, class_report, conf_matrix = imp_nn(X_train, y_train, X
print("accuracy:", accuracy)
print("auc score", auc_result)
print("\nclassification report:")
print(class_report)
print("\nconfusion matrix:")
print(conf_matrix)
```

```
Epoch 1/10
123/123 [==============================] - 1s 4ms/step - loss: 0.3456 - acc
uracy: 0.8749 - val_loss: 0.2975 - val_accuracy: 0.9038
Epoch 2/10
123/123 [==============================] - 0s 3ms/step - loss: 0.2804 - acc
uracy: 0.9059 - val_loss: 0.2755 - val_accuracy: 0.9038
Epoch 3/10
123/123 [==============================] - 0s 3ms/step - loss: 0.2598 - acc
uracy: 0.9062 - val_loss: 0.2571 - val_accuracy: 0.9038
Epoch 4/10
123/123 [==============================] - 0s 3ms/step - loss: 0.2427 - acc
uracy: 0.9064 - val_loss: 0.2424 - val_accuracy: 0.9061
Epoch 5/10
123/123 [==============================] - 0s 4ms/step - loss: 0.2289 - acc
uracy: 0.9087 - val_loss: 0.2310 - val_accuracy: 0.9072
Epoch 6/10
123/123 [==============================] - 0s 3ms/step - loss: 0.2176 - acc
uracy: 0.9128 - val_loss: 0.2225 - val_accuracy: 0.9140
Epoch 7/10
123/123 [==============================] - 1s 4ms/step - loss: 0.2093 - acc
uracy: 0.9171 - val_loss: 0.2157 - val_accuracy: 0.9152
Epoch 8/10
123/123 [==============================] - 0s 4ms/step - loss: 0.2026 - acc
uracy: 0.9194 - val_loss: 0.2113 - val_accuracy: 0.9186
Epoch 9/10
123/123 [==============================] - 0s 4ms/step - loss: 0.1978 - acc
uracy: 0.9234 - val_loss: 0.2083 - val_accuracy: 0.9219
Epoch 10/10
123/123 [==============================] - 1s 4ms/step - loss: 0.1937 - acc
uracy: 0.9252 - val_loss: 0.2046 - val_accuracy: 0.9197
4/4 [==============================] - 0s 4ms/step - loss: 0.1700 - accurac
y: 0.9293
Testing loss: 0.1700274497270584
Testing accuracy: 0.9292929172515869
```


The accuracy of the model
Train
Validation
Accuracy
0.92
0.91
0.90
0.89
0.88
0
2
4
6
8
Epoch


The loss of the model
Train
Validation
Loss
0.34
0.32
0.30
0.28
0.26
0.24
0.22
0.20
0
2
4
6
8
Epoch

```
4/4 [==============================] - 0s 3ms/step
accuracy: 0.9292929292929293
auc score 0.6818181818181819

classification report:
              precision    recall  f1-score   support

           0       1.00      0.36      0.53        11
           1       0.93      1.00      0.96        88

    accuracy                           0.93        99
   macro avg       0.96      0.68      0.75        99
weighted avg       0.93      0.93      0.91        99


confusion matrix:
[[ 4  7]
 [ 0 88]]
```

Summary

Based on the results of accuracy, precision, recall, f1 score, and AUC score, the differences between the two models are very small.

In terms of model complexity, data size, and training time, the Neural Network model is more complex than the Logistic Regression model, requires more data, and takes longer to train. Therefore, for this case where the data size is small and the analysis doesn't require complex computations for sentiment analysis, Logistic Regression would be a better choice.

In [ ]: