NLP_CW_F328685_George_Brown

March 15, 2024

1 Imports and set up

[1]:
```
!pip install -U nltk
!pip install spellchecker
!pip install pyspellchecker
import nltk; nltk.download('popular')
import pandas as pd
from google.colab import drive
import string
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
from collections import Counter
from os import listdir
from keras.preprocessing.text import Tokenizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
drive.mount('/content/drive')
import ipywidgets as widgets
from IPython.display import clear_output
clear_output()
```

2 Task 1

2.1 Clean data for vocab

The following code block processes the data for the vocabulary which is used for the models such as LSI. This is done first by reading the data as a data frame, extracts the reviews and then cleans the data using various tcehniques. These include: splitting by words , making all the tokens lowercase, removing punctuation, converting numbers to words i.e. '2' to 'two', removing stop words, removing short words, applying a spell corrector to re-spell misspelled words and then applied a lementizer to reduce the words to their root form. In addition, the icon ':)' was used a lot in the data however, instead of removing it, this was converted to the word 'happy'.

[2]:
```
import inflect
from spellchecker import SpellChecker
from nltk.stem import PorterStemmer, WordNetLemmatizer
```

```
def clean_data(txt):
    # split into tokens by white space
    tokens = txt.split()

    # convert :) to the word happy

    if ':)' in tokens:
        tokens[tokens.index(':)')] = 'happy'

    # convert each word to lowercase
    tokens = [word.lower() for word in tokens]

    # remove punctuation from each token
    table = str.maketrans('', '', string.punctuation)
    tokens = [w.translate(table) for w in tokens]

    # remove remaining tokens that are not alphabetic or numeric
    tokens = [word for word in tokens if word.isalpha() or word.isnumeric()]

    # convert numeric tokens to words
    p = inflect.engine()
    tokens = [p.number_to_words(word) if word.isnumeric() else word for word in␣
 ↪tokens]

    # remove stop words
    stop_words = set(stopwords.words('english'))
    tokens = [w for w in tokens if not w in stop_words]

    # remove short tokens
    tokens = [word for word in tokens if len(word) > 1]

    # Perform spelling correction and filter out None values
    spell = SpellChecker()
    tokens = [spell.correction(word) for word in tokens if spell.
 ↪correction(word) is not None]

    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]


    return tokens


def process_data(dataPath):
    df = pd.read_csv(dataPath)
```

```
    # get rid of duplicated rows
    df = df.drop_duplicates()

    # extract reviews
    ratings = df['Reviews']

    data = []
    # loop through the data, clean it and discard repeated data
    unique_lines_set = set()
    for line in ratings:
        if line not in unique_lines_set:
            cleaned_line = clean_data(line)
            data.append(cleaned_line)
            unique_lines_set.add(line)
            #print(cleaned_line)

    return data


# read data path
file1 = "/content/drive/MyDrive/Colab Notebooks/COP509cw/Datasets/
 ↪JewelleryReviewsLSA.csv"

data = process_data(file1)
```

2.2 Vocab

Next, a vocab was created by counting the occurance of words and then it filterring the vocabulary to only keep tokens with a minimum occurrence threshold of 2.

[10]:
```
all_words=[]
for word_list in data:
  for word in word_list:
    all_words.append(word)
print(len(all_words))
vocab = Counter()
vocab.update(all_words)
print(vocab.most_common(40))
```

```
3259
[('ring', 333), ('look', 69), ('like', 56), ('wear', 47), ('picture', 44),
('one', 39), ('love', 38), ('quality', 37), ('item', 36), ('beautiful', 31),
('would', 28), ('great', 25), ('nice', 24), ('bought', 20), ('stone', 19),
('bracelet', 19), ('gift', 18), ('diamond', 17), ('got', 17), ('silver', 17),
('small', 17), ('pretty', 17), ('color', 17), ('size', 16), ('time', 16),
('received', 16), ('price', 15), ('buy', 15), ('perfect', 15), ('recommend',
```

```
14), ('fit', 14), ("don't", 13), ('definitely', 13), ('wearing', 13),
('product', 13), ('looking', 13), ('little', 12), ('much', 12), ('also', 12),
('looked', 12)]
```

[11]:
```
# keep tokens with a min occurrence
min_occurane = 2
tokens = [k for k,c in vocab.items() if c >= min_occurane]
print(len(tokens))
vocab = set(tokens)
print(vocab)
```

```
433
{'believe', 'nothing', 'though', 'tongue', 'addiction', 'pick', 'solid',
'picture', 'princess', 'similar', 'stack', 'way', 'taste', 'receiving',
'description', 'dainty', 'necklace', 'recommend', 'week', 'always', 'purchase',
'suit', 'black', 'ring', 'joint', 'wedding', 'easily', 'right', 'wearing',
'purchased', 'sure', 'really', 'promptly', 'avenue', 'easy', 'casual',
'company', 'design', 'thing', 'quickly', 'ten', 'shell', 'fit', 'exactly',
'big', 'pretty', 'condition', 'polished', 'toe', 'tried', 'appearance', 'yet',
'bracelet', 'seemed', 'pendant', 'clear', 'worried', 'attractive', 'loved',
'already', 'bulkier', 'jeweler', 'bead', "doesn't", 'excellent', 'lost',
'quality', 'date', 'harm', 'new', 'first', 'year', 'buying', 'lover', 'many',
'hoped', 'something', 'pink', 'aware', 'timely', 'opened', 'claddagh', 'may',
'light', 'showed', 'product', 'get', 'next', 'day', 'cross', 'white', 'totally',
'birthday', 'sturdy', 'add', 'fused', 'arrived', 'one', 'given', 'otherwise',
'thin', 'smooth', 'girlfriend', 'stand', 'must', 'lot', 'shipping', 'imagine',
'flute', 'shipped', 'middle', 'perfectly', 'course', 'deal', 'second', 'back',
'within', 'gem', 'top', 'review', 'wear', 'fragile', 'store', 'expected',
'remember', 'came', 'thanks', 'advertised', 'wonderful', 'lip', 'enough',
'gift', 'high', 'present', 'every', 'catch', 'crack', 'happy', 'everyone',
'cut', 'type', 'platinum', 'finger', 'far', 'ear', 'flexibility', 'amazon',
'stone', 'misleading', 'awesome', 'close', 'wore', 'beautiful', 'clearly',
'smaller', 'wide', 'nice', 'shoulder', 'price', 'nine', 'truly', 'eye',
'attention', 'almost', 'bought', 'fact', 'fast', 'life', 'pinky', 'family',
'use', 'worth', 'delicate', 'scratch', 'blue', 'different', 'pleased',
'appreciate', 'thumb', 'say', 'ordering', 'customer', 'love', 'sits',
'returning', 'clean', 'occasion', 'dark', 'better', 'chip', 'sized', 'wife',
'large', 'work', 'vacation', 'i', 'perfect', 'completely', 'italic', 'forward',
'beauty', 'craftsmanship', 'enamel', 'size', 'someone', 'compliment', 'sent',
'topaz', 'comfortable', 'huge', 'lapel', 'visible', 'cool', 'help', 'also',
'manner', 'order', 'fan', 'go', 'low', 'ever', 'often', 'either', 'show',
'charm', 'heart', 'recently', 'actually', 'pewter', 'christian', 'man',
'earring', 'need', 'could', 'piece', 'described', 'well', 'without', 'small',
'jewelry', 'refund', 'site', 'anyone', 'cost', 'replacement', 'silver',
'expensive', 'unfortunately', 'cubic', 'together', 'around', 'simple',
'definitely', 'hold', 'seller', 'kept', 'turned', 'pictured', 'sparkle', 'seen',
'waste', 'flimsy', 'two', 'cute', 'bit', 'several', 'scratched', 'guess',
'time', 'anything', 'another', 'collection', 'notice', 'know', 'complaint',
```

```
'lovely', 'quick', 'extremely', 'poor', 'everything', 'gorgeous', 'sterling',
'four', 'missing', 'mine', 'prettier', 'spin', 'going', 'shopping', 'true',
'broken', 'everyday', 'wanted', 'last', 'expecting', 'side', 'much', 'noticed',
'absolutely', 'especially', 'away', 'long', 'flower', 'replace', 'little',
'old', 'my', 'people', 'made', 'toy', 'find', 'husband', 'cheap', 'ordered',
'although', 'regular', 'mail', 'glad', 'solitaire', 'wish', 'give', 'damaged',
'got', 'satisfied', 'thank', 'disappointed', 'make', "didn't", 'looked',
'expect', 'yellow', 'online', 'think', 'cant', 'style', 'cultic', 'part',
'impressed', 'barely', 'quite', 'color', 'detail', 'nicely', 'item', 'metal',
'amazing', 'suggest', 'look', 'amethyst', 'real', 'cannot', "don't", 'broke',
'beautifully', 'losing', 'turn', 'even', 'took', 'owned', 'hard', 'person',
'impossible', 'three', 'care', 'woman', 'sending', 'diamond', 'eve', 'gemstone',
'line', 'spend', 'daughter', 'catching', 'individual', 'zirconia', 'garnet',
'shine', 'overall', 'like', 'green', 'box', 'durable', 'decided', 'money',
'month', 'liked', 'fell', 'good', 'clasp', 'favorite', 'tell', 'stunning',
'friend', 'symbol', 'see', 'buy', 'head', 'cluster', 'however', 'receive',
'still', 'service', 'past', 'thought', 'since', 'anchor', 'le', 'great',
'shape', 'hand', 'returned', 'saw', 'want', 'please', 'thickness', 'resided',
'rosary', 'band', 'gold', "isn't", 'sparkling', 'comment', 'looking', 'night',
'purple', "couldn't", 'surprise', 'would', 'engagement', 'classy', 'received',
'elegant', 'ended'}
```

2.3 Clean data

Finally, the last part of the pre-processing was to clean the data for the models. Firstly, the data was read using a pandas data frame. Duplicate rows that were in the data were removed. The review data and associated id's were extracted and mapped using a dictionary so that this mapping could be used in the future for display purposes. Next, the data was cleaned the same way as before. In the data, it was noticed that there were duplicted reviews even though they had different ids. These duplicate reviews and ids were removed so that the remaining data and ids had no duplicates

[12]:
```
file1 = '/content/drive/MyDrive/Colab Notebooks/COP509cw/Datasets/
  ↪JewelleryReviewsLSA.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(file1)

# remove duplicates
newdf = df.drop_duplicates()

#ids column
ids = newdf['ID'].tolist()

# reviews column
data = newdf['Reviews'].tolist()

ratings = newdf['Ratings'].tolist()
```

```
# Mapping from id to raw data
id_to_raw_data = dict(zip(ids, data))

# Mapping from raw data to ratings

id_to_rating = dict(zip(data, ratings))

# Get unique lines and corresponding IDs
unique_data = []
unique_data_raw = []
unique_ids = []
unique_ratings = []

unique_lines_set = set()

# loop through data and remove duplicate ids and reviews
for line, id_value in zip(data, ids):
    if line not in unique_lines_set:
        clean = clean_data(line)
        cleaned_line = ' '.join(clean)
        unique_data.append(cleaned_line)
        unique_data_raw.append(line)
        unique_ids.append(id_value)
        unique_lines_set.add(line)

# mapping from clean data to id which could be used in future
id_to_clean_data = dict(zip(unique_ids, unique_data))

# mapping from clean data to raw data
clean_data_to_raw = dict(zip(unique_data, unique_data_raw))

ids = unique_ids
data = unique_data

# print to view cleaned data

#print(ids)
#print(data)
print(len(ids))
print(len(data))
```

```
184
184
```

Here is a word cloud to display the different words and relative importance for the data. Code source is show in comment below:

[13]:
```
import matplotlib.pyplot as plt


# source: https://towardsdatascience.com/
 ↪end-to-end-topic-modeling-in-python-latent-dirichlet-allocation-lda-35ce4ed6b3e0

# Import the wordcloud library
from wordcloud import WordCloud
# Join the different processed titles together.
long_string = ','.join(data)
# Create a WordCloud object
wordcloud = WordCloud(background_color="white", max_words=5000,␣
 ↪contour_width=3, contour_color='steelblue')
# Generate a word cloud
wordcloud.generate(long_string)
# Visualize the word c

# Visualize the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

3 Task 2 (LSI)

3.1 2a & 2C - Latent Semantic Indexing Model with Evaluation

The following code block uses a latent semantic indexing (LSI) model to retrieve the top 10 documents for 8 different queries.The data and queries are first transformed to a vectorized format using Tf-IDF. This algorithm represents the data in a high dimensional space and captures the importance of each term relative to the whole document. Then using Singular Value Decomposition (SVD), the data and query dimensions are reduced to the same dimenional space. The cosine similarity is then compared between the documents and the query. The 10 most similar docs are then retrieved. The precision, recall and f1 score is calculated by comparing the retrieved docs with the actual relevant docs. These evaluation methods were calculated across all queries and the average was calculated and presetnted with graphs

[14]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
import matplotlib.pyplot as plt


# This function processes the query by cleaning and applying TFIDF
def process_query(query, vocab, weightScheme):
    query = clean_data(query)
    line = ' '.join(query)
    vectorizer = CountVectorizer(vocabulary=vocab)
    transformer = TfidfTransformer(norm=weightScheme)
    encoded = transformer.fit_transform(vectorizer.fit_transform([line]))
    return encoded

# This function processes the data by appyling TFIDF
def vectorize_data(data, vocab, weightScheme):
    vectorizer = CountVectorizer(vocabulary=vocab)
    transformer = TfidfTransformer(norm= weightScheme)
    data_transformed = transformer.fit_transform(vectorizer.fit_transform(data))
    return data_transformed

# This function gets the queries needed to test this model
def get_queries(dataFrame):
    if len(dataFrame.columns) >= 17:
        # Extract values from columns 10 to 18 of the first row and convert to␣
 ↪strings
        values_list = [str(dataFrame.iloc[0, i]) for i in range(9, 17)]
        return values_list

# This function gets the relavent ids for each query
# for calculating precision, recall and f1
def get_relevant_ids(dataFrame):
```

```
    relevant_ids = []
    for column in dataFrame.columns[:8]:  # Loop through the first 8 columns
        column_values = dataFrame[column].dropna().astype(int).tolist()
        relevant_ids.append(column_values)

    return relevant_ids


# This function was created to calulate recall
def calculate_recall(rel, actual):
  number_of_relevant_ids = len(actual)
  recall = []
  total_relevant = 0
  for i in range(0, len(rel)):
    total_relevant += rel[i]

    r = total_relevant / number_of_relevant_ids
    recall.append(r)
  return recall

# This function was created to calculate precision
def calculate_precision(rel):
    precision = []
    total_relevant = 0
    for i in range(1, len(rel) + 1):
        total_relevant += rel[i - 1]

        if i != 0:
            p = total_relevant / i
        else:
            p = 0
        precision.append(p)

    return precision

# This function was created to calculate f1
def calculate_f1(precision, recall):
    F1 = []

    for i in range(len(precision)):


        if precision[i] + recall[i] != 0:
            f = 2 * (precision[i] * recall[i]) / (precision[i] + recall[i])
        else:
            f = 0
        F1.append(f)
```

```
    return F1


# Interplot Precision for standard Recall
def InterplotPrecision(p=0.1, Precision=None, Recall=None):

    if p >= 1.0:
        p = 0.9
    Mark = np.zeros(2)
    l = 0
    r = 0
    for i in range(len(Recall)):
        if Recall[i] >= p and Mark[0] == 0:
            l = i
            Mark[0] = 1
        if Recall[i] >= p + 0.1 and Mark[1] == 0:
        # if Recall[i] >= 1.0 and Mark[1] == 0:
            r = i
            Mark[1] = 1

    if (r < l) or (r==0 and l==0):

      y = Precision[-1]
    else:
      y = max(Precision[l:(r+1)])
    return y

# obtain y axis for R/P curve
def compute_RP_yaxis(Precision=None, Recall=None):
  y_axis = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

  for i in range(11):
    pInput = 0.1 * i
    y_axis[i] = InterplotPrecision(p=pInput, Precision=Precision, Recall=Recall)
  return y_axis

# get queries and process
file2 = '/content/drive/MyDrive/Colab Notebooks/COP509cw/Datasets/
 ↪JewelleryReviewsQueryRelevantID.csv'
querydf = pd.read_csv(file2)
queries = get_queries(querydf)

# get relevant ids
relevant_ids = get_relevant_ids(querydf)

# apply TFIDF on data
data_transformed = vectorize_data(data, vocab, '12')
```

```
# apply SVD on data
trunc_SVD_model = TruncatedSVD(n_components=50)
data_vectorized = trunc_SVD_model.fit_transform(data_transformed)

# mapping from id to vectorized document
id_to_vectorized = dict(zip(ids, data_vectorized))


# number of docs to retrieve
n = 10
j = 1

AllRecall = []
AllPrecision = []
AllF1measure = []

# Find similarities for each query
for query, query_relevant_ids in zip(queries, relevant_ids):

    # apply TFIDF on query
    processed_query = process_query(query, vocab, 'l2')
    # SVD on query
    transformed_query = trunc_SVD_model.transform(processed_query)
    # calculate similarities for data and query
    similarities = cosine_similarity(data_vectorized, transformed_query)
    # get indexes for top 10 documents
    indexes = np.argsort(similarities.flat)[-n:]

    # Retrieve the top n vectorized documents
    top_n_vectorized = data_vectorized[indexes]

    # Map the vectorized documents back to their corresponding IDs
    top_n_ids_lsi = [id for id, vectorized in id_to_vectorized.items() if␣
 ↪vectorized in top_n_vectorized]

    # Mark the relevant index
    re_mark = []
    for i in range(0, n):
        if top_n_ids_lsi[i] in query_relevant_ids:
            re_mark.append(1)
        else:
            re_mark.append(0)

    print("Query:{}".format(j))

    print("Top n document IDs: " + str(top_n_ids_lsi))
    print("Actual relevant ids:" + str(query_relevant_ids))
```

```
    print("Relevant", re_mark)

    precision = calculate_precision(re_mark)
    print("precision:", precision)

    recall = calculate_recall(re_mark, query_relevant_ids)
    print("Recall:", recall)

    F1 = calculate_f1(precision, recall)
    print("F1 Score", F1)

    print("-" * 200)

    # save
    AllRecall.append(recall)
    AllPrecision.append(precision)
    AllF1measure.append(F1)

    # plot R/P curve for each query
    x_axis = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
    y_axis = compute_RP_yaxis(Precision=precision, Recall=recall)
    plt.plot(x_axis, y_axis, '-o', label="Query %d" % (j))

    j += 1

# Put in numpy array
AllRecall = np.array(AllRecall)
AllPrecision = np.array(AllPrecision)
AllF1measure = np.array(AllF1measure)

# Compute average Recall, average Precision, average F1-measure for all queries
AveRecall = np.mean(AllRecall, axis=0)
AvePrecision = np.mean(AllPrecision, axis=0)
AveF1measure = np.mean(AllF1measure, axis=0)

# Calculate standard deviation
StdRecall = np.std(AllRecall, axis=0)
StdPrecision = np.std(AllPrecision, axis=0)
StdF1measure = np.std(AllF1measure, axis=0)

print("\nAverage Recall, average Precision, average F1-measure: ")
print("average Recall@1~10: ", np.around(AveRecall[:10],2))
print("average Precision@1~10: ", np.around(AvePrecision[:10],2))
print("average F1measure@1~10: ", np.around(AveF1measure[:10],2))

print("\nStandard Deviation Recall, Precision, F1-measure: ")
```

```
print("Standard Deviation Recall@1~10: ", np.around(StdRecall[:10], 2))
print("Standard Deviation Precision@1~10: ", np.around(StdPrecision[:10], 2))
print("Standard Deviation F1measure@1~10: ", np.around(StdF1measure[:10], 2))

# Display the overall plot after all queries

plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Standard Recall/Precision Curves')
plt.legend()
plt.show()


# plot average R/P curve

x_axis = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
y_axis = compute_RP_yaxis(Precision=AvePrecision, Recall=AveRecall)
plt.plot(x_axis, y_axis, '-bo', color="blue", label="Average")
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel('average Recall')
plt.ylabel('average Precision')
plt.title('Standard Average Recall/Precision Curves')
plt.legend()
plt.show()
```

```
Query:1
Top n document IDs: [6522, 58481, 26246, 2033, 48779, 34523, 49525, 41876,
17309, 17273]
Actual relevant ids:[36164, 58481, 26246, 2033, 48779, 34523, 9726, 56494,
49525, 45278, 35694, 41876, 17309, 11135, 17273, 11247]
Relevant [0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
precision: [0.0, 0.5, 0.6666666666666666, 0.75, 0.8, 0.8333333333333334,
0.8571428571428571, 0.875, 0.8888888888888888, 0.9]
Recall: [0.0, 0.0625, 0.125, 0.1875, 0.25, 0.3125, 0.375, 0.4375, 0.5, 0.5625]
F1 Score [0, 0.1111111111111111, 0.21052631578947367, 0.3, 0.38095238095238093,
0.45454545454545453, 0.5217391304347825, 0.5833333333333334, 0.64,
0.6923076923076923]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Query:2
Top n document IDs: [51907, 57123, 25299, 55017, 7432, 2114, 40871, 13373, 642,
56865]
Actual relevant ids:[57123, 25299, 55017, 7432, 2114, 40871]
```

```
Relevant [0, 1, 1, 1, 1, 1, 1, 0, 0, 0]
precision: [0.0, 0.5, 0.6666666666666666, 0.75, 0.8, 0.8333333333333334,
0.8571428571428571, 0.75, 0.6666666666666666, 0.6]
Recall: [0.0, 0.16666666666666666, 0.3333333333333333, 0.5, 0.6666666666666666,
0.8333333333333334, 1.0, 1.0, 1.0, 1.0]
F1 Score [0, 0.25, 0.4444444444444444, 0.6, 0.7272727272727272,
0.8333333333333334, 0.923076923076923, 0.8571428571428571, 0.8,
0.7499999999999999]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------
Query:3
Top n document IDs: [11087, 56342, 45856, 4375, 22058, 9726, 33251, 2780, 13373,
3494]
Actual relevant ids:[33251, 17304, 50019, 27679, 6158, 22408, 29722, 36677,
2780, 17944, 19944, 31657, 52867, 49216]
Relevant [0, 0, 0, 0, 0, 0, 1, 1, 0, 0]
precision: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.14285714285714285, 0.25,
0.2222222222222222, 0.2]
Recall: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.07142857142857142, 0.14285714285714285,
0.14285714285714285, 0.14285714285714285]
F1 Score [0, 0, 0, 0, 0, 0, 0.09523809523809523, 0.18181818181818182,
0.17391304347826086, 0.16666666666666666]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------
Query:4
Top n document IDs: [54548, 535, 33746, 37486, 30640, 19852, 26535, 21070,
53660, 44126]
Actual relevant ids:[40373, 28648, 37486, 30640, 2131, 19852, 2134, 36585,
26535, 51474, 21070, 56330, 53660, 44126]
Relevant [0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
precision: [0.0, 0.0, 0.0, 0.25, 0.4, 0.5, 0.5714285714285714, 0.625,
0.6666666666666666, 0.7]
Recall: [0.0, 0.0, 0.0, 0.07142857142857142, 0.14285714285714285,
0.21428571428571427, 0.2857142857142857, 0.35714285714285715,
0.42857142857142855, 0.5]
F1 Score [0, 0, 0, 0.11111111111111112, 0.21052631578947364, 0.3,
0.38095238095238093, 0.45454545454545453, 0.5217391304347826,
0.5833333333333334]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------
Query:5
Top n document IDs: [1816, 265, 36727, 45548, 4375, 51907, 22058, 13373, 17607,
33571]
Actual relevant ids:[13373, 17607, 41459, 54748, 33571]
Relevant [0, 0, 0, 0, 0, 0, 0, 1, 1, 1]
```

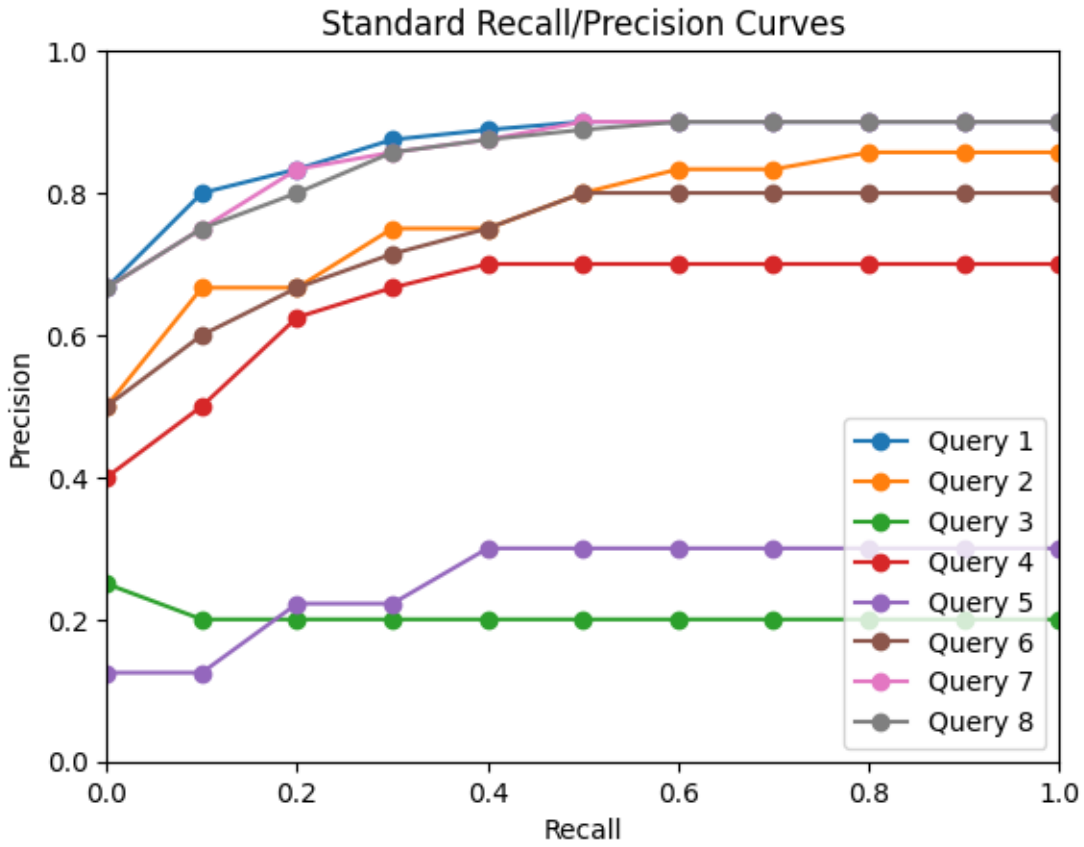
precision: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.125, 0.2222222222222222, 0.3]
Recall: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 0.4, 0.6]
F1 Score [0, 0, 0, 0, 0, 0, 0, 0.15384615384615385, 0.2857142857142857, 0.4]

Query:6
Top n document IDs: [42077, 10612, 45860, 46500, 27474, 41319, 39932, 50197, 8341, 52375]
Actual relevant ids:[45860, 46500, 27474, 43945, 52837, 12358, 41319, 39932, 45146, 50197, 8341, 52375]
Relevant [0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
precision: [0.0, 0.0, 0.3333333333333333, 0.5, 0.6, 0.6666666666666666, 0.7142857142857143, 0.75, 0.7777777777777778, 0.8]
Recall: [0.0, 0.0, 0.08333333333333333, 0.16666666666666666, 0.25, 0.3333333333333333, 0.4166666666666667, 0.5, 0.5833333333333334, 0.6666666666666666]
F1 Score [0, 0, 0.13333333333333333, 0.25, 0.35294117647058826, 0.4444444444444444, 0.5263157894736842, 0.6, 0.6666666666666666, 0.7272727272727272]

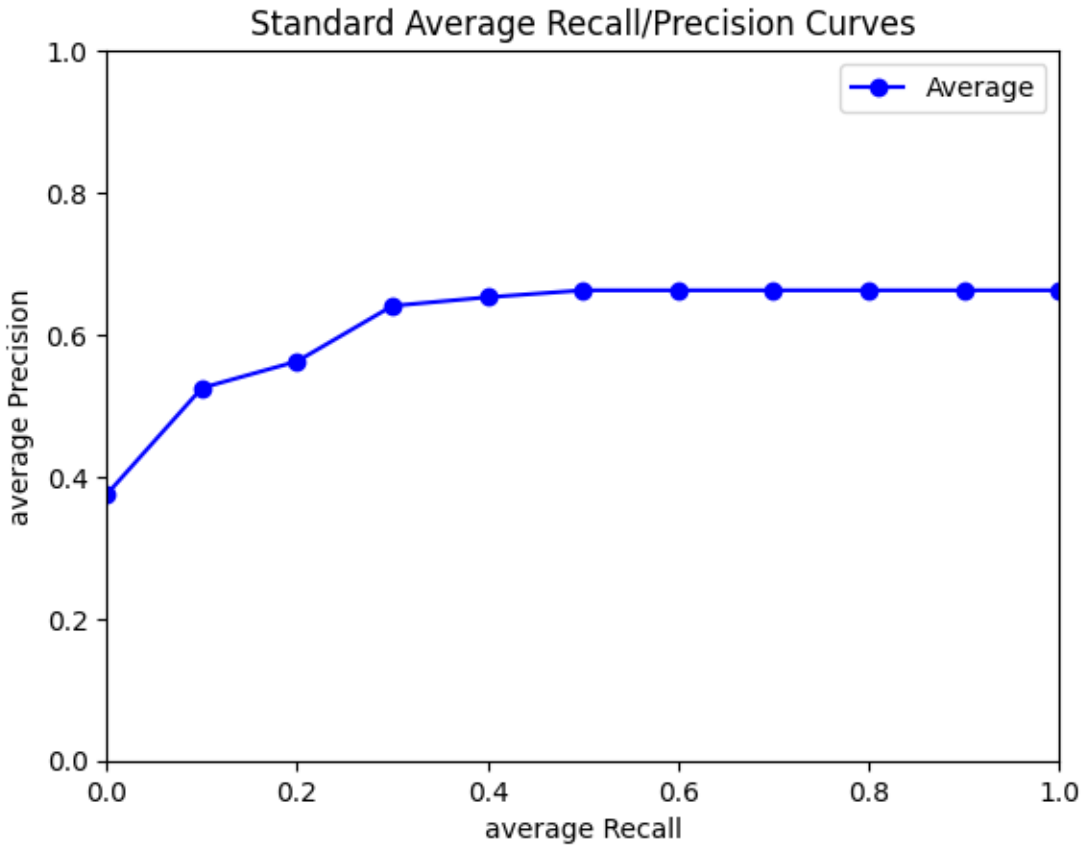
Query:7
Top n document IDs: [32767, 209, 216, 47345, 38637, 7110, 51356, 36165, 943, 37864]
Actual relevant ids:[209, 28542, 216, 47345, 11356, 33632, 38637, 7110, 6649, 51356, 44358, 36165, 943, 37864]
Relevant [0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
precision: [0.0, 0.5, 0.6666666666666666, 0.75, 0.8, 0.8333333333333334, 0.8571428571428571, 0.875, 0.8888888888888888, 0.9]
Recall: [0.0, 0.07142857142857142, 0.14285714285714285, 0.21428571428571427, 0.2857142857142857, 0.35714285714285715, 0.42857142857142855, 0.5, 0.5714285714285714, 0.6428571428571429]
F1 Score [0, 0.125, 0.23529411764705882, 0.3333333333333333, 0.4210526315789473, 0.5, 0.5714285714285714, 0.6363636363636364, 0.6956521739130435, 0.75]

Query:8
Top n document IDs: [25299, 642, 10642, 37794, 45518, 3494, 735, 41872, 53409, 56865]
Actual relevant ids:[642, 10642, 37794, 45518, 3494, 735, 10037, 41872, 28542, 53409, 56865, 44489, 44490]
Relevant [0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
precision: [0.0, 0.5, 0.6666666666666666, 0.75, 0.8, 0.8333333333333334, 0.8571428571428571, 0.875, 0.8888888888888888, 0.9]
Recall: [0.0, 0.07692307692307693, 0.15384615384615385, 0.23076923076923078,

```
0.3076923076923077, 0.38461538461538464, 0.46153846153846156,
0.5384615384615384, 0.6153846153846154, 0.6923076923076923]
F1 Score [0, 0.13333333333333336, 0.25, 0.3529411764705882, 0.4444444444444444,
0.5263157894736842, 0.6, 0.6666666666666667, 0.7272727272727274,
0.7826086956521738]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------

Average Recall, average Precision, average F1-measure:
average Recall@1~10:  [0.   0.05 0.1  0.17 0.24 0.3  0.38 0.46 0.53 0.6 ]
average Precision@1~10:  [0.   0.25 0.37 0.47 0.52 0.56 0.61 0.64 0.65 0.66]
average F1measure@1~10:  [0.   0.08 0.16 0.24 0.32 0.38 0.45 0.52 0.56 0.61]

Standard Deviation Recall, Precision, F1-measure:
Standard Deviation Recall@1~10:  [0.   0.06 0.11 0.15 0.2  0.25 0.28 0.25 0.23
0.22]
Standard Deviation Precision@1~10:  [0.   0.25 0.31 0.32 0.33 0.34 0.33 0.28
0.26 0.26]
Standard Deviation F1measure@1~10:  [0.   0.09 0.15 0.19 0.23 0.26 0.28 0.23
0.21 0.2 ]
```



```
<ipython-input-14-a61d72392951>:239: UserWarning: color is redundantly defined
by the 'color' keyword argument and the fmt string "-bo" (-> color='b'). The
keyword argument will take precedence.
  plt.plot(x_axis, y_axis, '-bo', color="blue", label="Average")
```



3.2 2B & 2C - Model Tuning with Evaluation

This code block optimizes the model by looping through all the different combinations of component values for the SVD dimensions with the 'norm' value of scikit learns TFIDF parameter. The average precison, recall and f1 is calculated and plotted. The best combination of oparameters is determined by the best average f1 score.

[15]:
```
from IPython.display import clear_output

# get queries
file2 = '/content/drive/MyDrive/Colab Notebooks/COP509cw/Datasets/
  ↪JewelleryReviewsQueryRelevantID.csv'
querydf = pd.read_csv(file2)
queries = get_queries(querydf)
```

```
# get relevant ids

relevant_ids = get_relevant_ids(querydf)

component_values = [10, 20, 30, 40, 50]
norm_schemes = ['l1', 'l2']

a = 1

best_avg_f1 = 0.0
best_norm = ''
best_n_comp = 0
tuned_model = 0

# Experiment with different parameters
for n_components in component_values:
    for norm in norm_schemes:

        # SVD on data
        data_transformed = vectorize_data(data, vocab, norm)
        trunc_SVD_model = TruncatedSVD(n_components=n_components)
        data_vectorized = trunc_SVD_model.fit_transform(data_transformed)


        # mapping from id to vextorized document
        id_to_vectorized = dict(zip(ids, data_vectorized))


        n = 10
        j = 1

        AllRecall = []
        AllPrecision = []
        AllF1measure = []

        # Find similarities for each query
        for query, query_relevant_ids in zip(queries, relevant_ids):
            processed_query = process_query(query, vocab, norm)

            transformed_query = trunc_SVD_model.transform(processed_query)
            similarities = cosine_similarity(data_vectorized, transformed_query)
            indexes = np.argsort(similarities.flat)[-n:]

            # Retrieve the top n vectorized documents
            top_n_vectorized = data_vectorized[indexes]
```

```
            # Map the vectorized documents back to their corresponding IDs
            top_n_ids_lsi = [id for id, vectorized in id_to_vectorized.items()␣
↪if vectorized in top_n_vectorized]

            # Mark the relevant index
            re_mark = []
            for i in range(0, n):
                if top_n_ids_lsi[i] in query_relevant_ids:
                    re_mark.append(1)
                else:
                    re_mark.append(0)


            precision = calculate_precision(re_mark)
            recall = calculate_recall(re_mark, query_relevant_ids)
            F1 = calculate_f1(precision, recall)

              # save
            AllRecall.append(recall)
            AllPrecision.append(precision)
            AllF1measure.append(F1)


            j +=1

        # compute average Recall, average Precision, average F1-measure

        AllRecall = np.array(AllRecall)
        AllPrecision = np.array(AllPrecision)
        AllF1measure = np.array(AllF1measure)

        # Compute average Recall, average Precision, average F1-measure for all␣
↪queries
        AveRecall = np.mean(AllRecall, axis=0)
        AvePrecision = np.mean(AllPrecision, axis=0)
        AveF1measure = np.mean(AllF1measure, axis=0)

        # Calculate standard deviation
        StdRecall = np.std(AllRecall, axis=0)
        StdPrecision = np.std(AllPrecision, axis=0)
        StdF1measure = np.std(AllF1measure, axis=0)

        print("Tune setting {}: \nnumber of components - {}, \nNorm scheme -␣
↪{}".format(a,n_components,norm ))
        print("\nAverage Recall, average Precision, average F1-measure: ")
        print("average Recall@1~10: ", np.around(AveRecall[:10],2))
        print("average Precision@1~10: ", np.around(AvePrecision[:10],2))
```

```
        print("average F1measure@1~10: ", np.around(AveF1measure[:10],2))

        print("\nStandard Deviation of Recall, Precision, F1-measure: ")
        print("Std Dev of Recall@1~10: ", np.around(StdRecall[:10], 2))
        print("Std Dev of Precision@1~10: ", np.around(StdPrecision[:10], 2))
        print("Std Dev of F1measure@1~10: ", np.around(StdF1measure[:10], 2))

        print('-'* 200)

        # Update best results if the current iteration is better
        if np.sum(AveF1measure) > best_avg_f1:
            best_avg_f1 = np.sum(AveF1measure)
            best_n_comp = n_components
            best_norm = norm
            tuned_model = a


        # plot R/P curve for each query
        x_axis = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
        y_axis = compute_RP_yaxis(Precision=AvePrecision, Recall=AveRecall)
        plt.plot(x_axis, y_axis, '-o', label="Tune %d" % (a))

        a += 1


# Save the best top N IDs
print("\nBest Tuned model: {}".format(tuned_model))
print("Number of components: {}, Norm scheme: {}".format(best_n_comp,␣
 ↪best_norm))
print("Best Average F1-measure:", best_avg_f1)


plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('AverGE Recall/Precision Curves for different tune settings')
plt.legend()
plt.show()
plt.savefig("tuned_models")
```

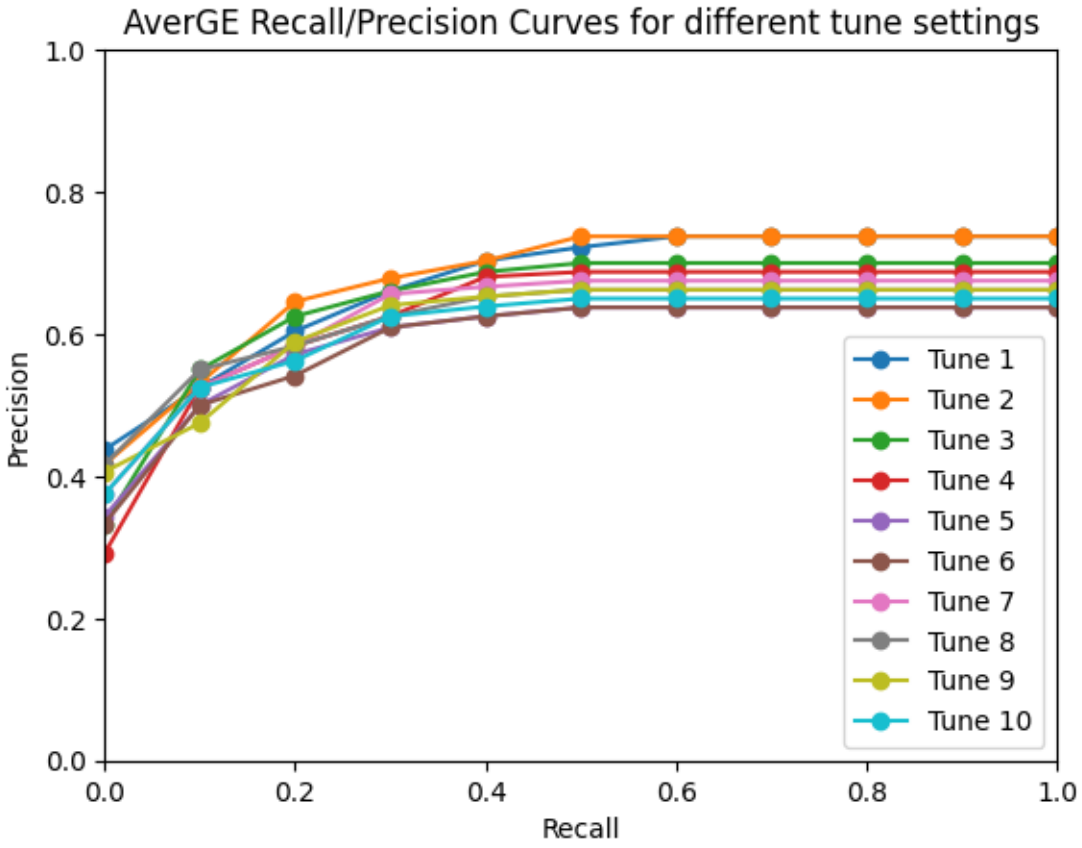
```
Tune setting 1:
number of components - 10,
Norm scheme - l1

Average Recall, average Precision, average F1-measure:
average Recall@1~10:  [0.01 0.02 0.08 0.15 0.23 0.33 0.43 0.53 0.61 0.69]
```

```
average Precision@1~10:  [0.12 0.12 0.29 0.44 0.52 0.6  0.66 0.7  0.72 0.74]
average F1measure@1~10:  [0.02 0.03 0.12 0.22 0.31 0.41 0.5  0.58 0.63 0.68]

Standard Deviation of Recall, Precision, F1-measure:
Std Dev of Recall@1~10:  [0.03 0.05 0.08 0.11 0.14 0.15 0.17 0.2  0.19 0.2 ]
Std Dev of Precision@1~10:  [0.33 0.33 0.31 0.27 0.26 0.22 0.19 0.16 0.15 0.14]
Std Dev of F1measure@1~10:  [0.05 0.09 0.12 0.14 0.17 0.15 0.14 0.14 0.11 0.08]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Tune setting 2:
number of components - 10,
Norm scheme - l2

Average Recall, average Precision, average F1-measure:
average Recall@1~10:  [0.03 0.08 0.13 0.2  0.28 0.35 0.43 0.51 0.59 0.67]
average Precision@1~10:  [0.25 0.38 0.42 0.53 0.6  0.65 0.68 0.7  0.72 0.74]
average F1measure@1~10:  [0.05 0.13 0.19 0.28 0.37 0.44 0.51 0.57 0.63 0.68]

Standard Deviation of Recall, Precision, F1-measure:
Std Dev of Recall@1~10:  [0.06 0.11 0.16 0.2  0.24 0.28 0.24 0.2  0.18 0.17]
Std Dev of Precision@1~10:  [0.43 0.41 0.43 0.36 0.33 0.32 0.26 0.22 0.19 0.18]
Std Dev of F1measure@1~10:  [0.1  0.17 0.23 0.24 0.26 0.28 0.22 0.17 0.14 0.11]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Tune setting 3:
number of components - 20,
Norm scheme - l1

Average Recall, average Precision, average F1-measure:
average Recall@1~10:  [0.02 0.06 0.11 0.18 0.26 0.36 0.44 0.52 0.59 0.66]
average Precision@1~10:  [0.12 0.25 0.33 0.47 0.55 0.62 0.66 0.69 0.69 0.7 ]
average F1measure@1~10:  [0.04 0.09 0.16 0.25 0.34 0.44 0.51 0.57 0.61 0.65]

Standard Deviation of Recall, Precision, F1-measure:
Std Dev of Recall@1~10:  [0.06 0.11 0.16 0.2  0.23 0.25 0.22 0.2  0.2  0.22]
Std Dev of Precision@1~10:  [0.33 0.35 0.37 0.32 0.3  0.25 0.19 0.15 0.14 0.15]
Std Dev of F1measure@1~10:  [0.09 0.16 0.22 0.23 0.25 0.24 0.18 0.13 0.11 0.11]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Tune setting 4:
number of components - 20,
Norm scheme - l2

Average Recall, average Precision, average F1-measure:
average Recall@1~10:  [0.03 0.06 0.1  0.18 0.25 0.33 0.41 0.49 0.57 0.64]
```

```
average Precision@1~10:  [0.25 0.25 0.29 0.44 0.52 0.58 0.62 0.66 0.68 0.69]
average F1measure@1~10:  [0.05 0.1  0.14 0.24 0.33 0.4  0.47 0.54 0.59 0.63]

Standard Deviation of Recall, Precision, F1-measure:
Std Dev of Recall@1~10:  [0.06 0.11 0.17 0.2  0.24 0.28 0.24 0.21 0.19 0.19]
Std Dev of Precision@1~10:  [0.43 0.43 0.42 0.35 0.32 0.3  0.24 0.2  0.17 0.16]
Std Dev of F1measure@1~10:  [0.1  0.18 0.23 0.25 0.26 0.28 0.22 0.17 0.13 0.12]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Tune setting 5:
number of components - 30,
Norm scheme - l1

Average Recall, average Precision, average F1-measure:
average Recall@1~10:  [0.   0.03 0.08 0.13 0.2  0.28 0.38 0.46 0.53 0.6 ]
average Precision@1~10:  [0.   0.12 0.25 0.34 0.42 0.5  0.57 0.61 0.62 0.64]
average F1measure@1~10:  [0.   0.05 0.11 0.18 0.26 0.34 0.44 0.5  0.55 0.59]

Standard Deviation of Recall, Precision, F1-measure:
Std Dev of Recall@1~10:  [0.   0.06 0.11 0.16 0.2  0.24 0.25 0.22 0.21 0.22]
Std Dev of Precision@1~10:  [0.   0.22 0.28 0.3  0.31 0.29 0.25 0.2  0.19 0.19]
Std Dev of F1measure@1~10:  [0.   0.09 0.15 0.2  0.23 0.24 0.23 0.18 0.16 0.15]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Tune setting 6:
number of components - 30,
Norm scheme - l2

Average Recall, average Precision, average F1-measure:
average Recall@1~10:  [0.02 0.05 0.11 0.18 0.24 0.31 0.36 0.44 0.51 0.58]
average Precision@1~10:  [0.12 0.19 0.33 0.44 0.5  0.54 0.57 0.61 0.62 0.64]
average F1measure@1~10:  [0.04 0.08 0.16 0.24 0.31 0.38 0.43 0.49 0.54 0.59]

Standard Deviation of Recall, Precision, F1-measure:
Std Dev of Recall@1~10:  [0.06 0.11 0.16 0.2  0.25 0.29 0.28 0.25 0.23 0.22]
Std Dev of Precision@1~10:  [0.33 0.35 0.33 0.32 0.33 0.34 0.3  0.25 0.24 0.24]
Std Dev of F1measure@1~10:  [0.09 0.16 0.21 0.24 0.27 0.3  0.27 0.22 0.2  0.19]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Tune setting 7:
number of components - 40,
Norm scheme - l1

Average Recall, average Precision, average F1-measure:
average Recall@1~10:  [0.01 0.06 0.1  0.17 0.24 0.31 0.39 0.47 0.54 0.61]
```

```
average Precision@1~10:  [0.12 0.31 0.38 0.47 0.52 0.58 0.62 0.66 0.67 0.68]
average F1measure@1~10:  [0.02 0.09 0.16 0.24 0.32 0.39 0.46 0.53 0.57 0.62]

Standard Deviation of Recall, Precision, F1-measure:
Std Dev of Recall@1~10:  [0.03 0.07 0.12 0.16 0.2  0.24 0.28 0.24 0.22 0.21]
Std Dev of Precision@1~10:  [0.33 0.35 0.39 0.36 0.36 0.33 0.32 0.27 0.25 0.25]
Std Dev of F1measure@1~10:  [0.05 0.11 0.17 0.21 0.24 0.25 0.27 0.22 0.2  0.19]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Tune setting 8:
number of components - 40,
Norm scheme - l2

Average Recall, average Precision, average F1-measure:
average Recall@1~10:  [0.02 0.07 0.13 0.19 0.26 0.33 0.37 0.45 0.53 0.6 ]
average Precision@1~10:  [0.12 0.31 0.42 0.5  0.55 0.58 0.59 0.62 0.65 0.66]
average F1measure@1~10:  [0.04 0.11 0.19 0.27 0.34 0.4  0.44 0.51 0.56 0.61]

Standard Deviation of Recall, Precision, F1-measure:
Std Dev of Recall@1~10:  [0.06 0.11 0.15 0.2  0.25 0.29 0.29 0.26 0.23 0.22]
Std Dev of Precision@1~10:  [0.33 0.35 0.36 0.35 0.36 0.36 0.35 0.3  0.26 0.26]
Std Dev of F1measure@1~10:  [0.09 0.16 0.21 0.24 0.27 0.3  0.29 0.24 0.21 0.2 ]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Tune setting 9:
number of components - 50,
Norm scheme - l1

Average Recall, average Precision, average F1-measure:
average Recall@1~10:  [0.   0.03 0.07 0.14 0.21 0.28 0.36 0.46 0.53 0.6 ]
average Precision@1~10:  [0.   0.19 0.29 0.41 0.48 0.54 0.59 0.64 0.65 0.66]
average F1measure@1~10:  [0.   0.05 0.11 0.2  0.28 0.36 0.43 0.52 0.56 0.61]

Standard Deviation of Recall, Precision, F1-measure:
Std Dev of Recall@1~10:  [0.   0.03 0.07 0.11 0.16 0.19 0.23 0.23 0.21 0.2 ]
Std Dev of Precision@1~10:  [0.   0.24 0.31 0.3  0.32 0.3  0.29 0.25 0.24 0.23]
Std Dev of F1measure@1~10:  [0.   0.06 0.12 0.15 0.19 0.21 0.23 0.21 0.19 0.18]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Tune setting 10:
number of components - 50,
Norm scheme - l2

Average Recall, average Precision, average F1-measure:
average Recall@1~10:  [0.02 0.06 0.12 0.18 0.25 0.32 0.37 0.45 0.52 0.59]
```

```
average Precision@1~10:  [0.12 0.25 0.37 0.47 0.52 0.56 0.59 0.62 0.64 0.65]
average F1measure@1~10:  [0.04 0.09 0.17 0.26 0.33 0.39 0.44 0.51 0.55 0.6 ]

Standard Deviation of Recall, Precision, F1-measure:
Std Dev of Recall@1~10:  [0.06 0.11 0.16 0.2  0.25 0.29 0.28 0.25 0.23 0.22]
Std Dev of Precision@1~10:  [0.33 0.35 0.35 0.34 0.35 0.35 0.31 0.27 0.25 0.25]
Std Dev of F1measure@1~10:  [0.09 0.16 0.21 0.24 0.27 0.3  0.27 0.23 0.21 0.2 ]
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------

Best Tuned model: 2
Number of components: 10, Norm scheme: l2
Best Average F1-measure: 3.8483675529388277
```




```
<Figure size 640x480 with 0 Axes>
```

3.3 2B & 2C - Best Model

This code block then re-runs that optimized model to show the precison, recall and f1 for the different queries

[16]:
```
# get queries
file2 = '/content/drive/MyDrive/Colab Notebooks/COP509cw/Datasets/
 ↪JewelleryReviewsQueryRelevantID.csv'
querydf = pd.read_csv(file2)
queries = get_queries(querydf)

# get relevant ids

relevant_ids = get_relevant_ids(querydf)

# SVD on data
data_transformed = vectorize_data(data, vocab, best_norm)
trunc_SVD_model = TruncatedSVD(n_components=best_n_comp)
data_vectorized = trunc_SVD_model.fit_transform(data_transformed)


# mapping from id to vextorized document
id_to_vectorized = dict(zip(ids, data_vectorized))


n = 10
j = 1


AllRecall = []
AllPrecision = []
AllF1measure = []



# Find similarities for each query
for query, query_relevant_ids in zip(queries, relevant_ids):
    processed_query = process_query(query, vocab, best_norm)

    transformed_query = trunc_SVD_model.transform(processed_query)
    similarities = cosine_similarity(data_vectorized, transformed_query)
    indexes = np.argsort(similarities.flat)[-n:]

    # Retrieve the top n vectorized documents
    top_n_vectorized = data_vectorized[indexes]

    # Map the vectorized documents back to their corresponding IDs
    top_n_ids_lsi = [id for id, vectorized in id_to_vectorized.items() if␣
 ↪vectorized in top_n_vectorized]

    # Mark the relevant index
    re_mark = []
```

```
    for i in range(0, n):
        if top_n_ids_lsi[i] in query_relevant_ids:
            re_mark.append(1)
        else:
            re_mark.append(0)


    print("Query:{}".format(j))

    print("Top n document IDs: " + str(top_n_ids_lsi))
    print("Actual relevant ids:" + str(query_relevant_ids))

    print("Relevant", re_mark)

    precision = calculate_precision(re_mark)
    print("precision:", precision)

    recall = calculate_recall(re_mark, query_relevant_ids)
    print("Recall:", recall)

    F1 = calculate_f1(precision, recall)
    print("F1 Score", F1)

    print("-" * 200)

    # save
    AllRecall.append(recall)
    AllPrecision.append(precision)
    AllF1measure.append(F1)

    # plot R/P curve for each query
    x_axis = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
    y_axis = compute_RP_yaxis(Precision=precision, Recall=recall)
    plt.plot(x_axis, y_axis, '-o', label="Query %d" % (j))

    j += 1



AllRecall = np.array(AllRecall)
AllPrecision = np.array(AllPrecision)
AllF1measure = np.array(AllF1measure)

# compute average Recall, average Precision, average F1-measure
BestAveRecall = np.mean(AllRecall, axis=0)
BestAvePrecision = np.mean(AllPrecision, axis=0)
```

```
BestAveF1measure = np.mean(AllF1measure, axis=0)

# Calculate standard deviation
BestStdRecall = np.std(AllRecall, axis=0)
BestStdPrecision = np.std(AllPrecision, axis=0)
BestStdF1measure = np.std(AllF1measure, axis=0)

print("\nAverage Recall, average Precision, average F1-measure: ")
print("average Recall@1~10: ", np.around(BestAveRecall[:10],2))
print("average Precision@1~10: ", np.around(BestAvePrecision[:10],2))
print("average F1measure@1~10: ", np.around(BestAveF1measure[:10],2))

print("\nStandard Deviation of Recall, Precision, F1-measure: ")
print("Std Dev of Recall@1~10: ", np.around(BestStdRecall[:10], 2))
print("Std Dev of Precision@1~10: ", np.around(BestStdPrecision[:10], 2))
print("Std Dev of F1measure@1~10: ", np.around(BestStdF1measure[:10], 2))

# Display the overall plot after all queries

plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Standard Recall/Precision Curves')
plt.legend()
plt.savefig("best_model_pr.png")
plt.show()



# plot average R/P curve

x_axis = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
y_axis = compute_RP_yaxis(Precision=BestAvePrecision, Recall=BestAveRecall)
plt.plot(x_axis, y_axis, '-bo', color="blue", label="Average")
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel('average Recall')
plt.ylabel('average Precision')
plt.title('Standard Average Recall/Precision Curves')
plt.legend()
plt.savefig("best_model_pr_average.png")
plt.show()
```

```
Query:1
Top n document IDs: [48216, 44591, 36164, 58481, 26246, 2033, 48779, 34523,
45278, 41876]
```

```
Actual relevant ids:[36164, 58481, 26246, 2033, 48779, 34523, 9726, 56494,
49525, 45278, 35694, 41876, 17309, 11135, 17273, 11247]
Relevant [0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
precision: [0.0, 0.0, 0.3333333333333333, 0.5, 0.6, 0.6666666666666666,
0.7142857142857143, 0.75, 0.7777777777777778, 0.8]
Recall: [0.0, 0.0, 0.0625, 0.125, 0.1875, 0.25, 0.3125, 0.375, 0.4375, 0.5]
F1 Score [0, 0, 0.10526315789473684, 0.2, 0.2857142857142857,
0.36363636363636365, 0.43478260869565216, 0.5, 0.56, 0.6153846153846154]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Query:2
Top n document IDs: [57123, 25299, 55017, 7432, 2114, 40871, 13373, 54748, 642,
56865]
Actual relevant ids:[57123, 25299, 55017, 7432, 2114, 40871]
Relevant [1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
precision: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.8571428571428571, 0.75,
0.6666666666666666, 0.6]
Recall: [0.16666666666666666, 0.3333333333333333, 0.5, 0.6666666666666666,
0.8333333333333334, 1.0, 1.0, 1.0, 1.0, 1.0]
F1 Score [0.2857142857142857, 0.5, 0.6666666666666666, 0.8, 0.9090909090909091,
1.0, 0.923076923076923, 0.8571428571428571, 0.8, 0.7499999999999999]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Query:3
Top n document IDs: [22058, 33251, 17304, 50019, 27679, 29722, 2780, 17944,
31657, 49216]
Actual relevant ids:[33251, 17304, 50019, 27679, 6158, 22408, 29722, 36677,
2780, 17944, 19944, 31657, 52867, 49216]
Relevant [0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
precision: [0.0, 0.5, 0.6666666666666666, 0.75, 0.8, 0.8333333333333334,
0.8571428571428571, 0.875, 0.8888888888888888, 0.9]
Recall: [0.0, 0.07142857142857142, 0.14285714285714285, 0.21428571428571427,
0.2857142857142857, 0.35714285714285715, 0.42857142857142855, 0.5,
0.5714285714285714, 0.6428571428571429]
F1 Score [0, 0.125, 0.23529411764705882, 0.3333333333333333, 0.4210526315789473,
0.5, 0.5714285714285714, 0.6363636363636364, 0.6956521739130435, 0.75]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Query:4
Top n document IDs: [54548, 535, 33746, 37486, 30640, 19852, 26535, 21070,
53660, 44126]
Actual relevant ids:[40373, 28648, 37486, 30640, 2131, 19852, 2134, 36585,
26535, 51474, 21070, 56330, 53660, 44126]
Relevant [0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
precision: [0.0, 0.0, 0.0, 0.25, 0.4, 0.5, 0.5714285714285714, 0.625,
```

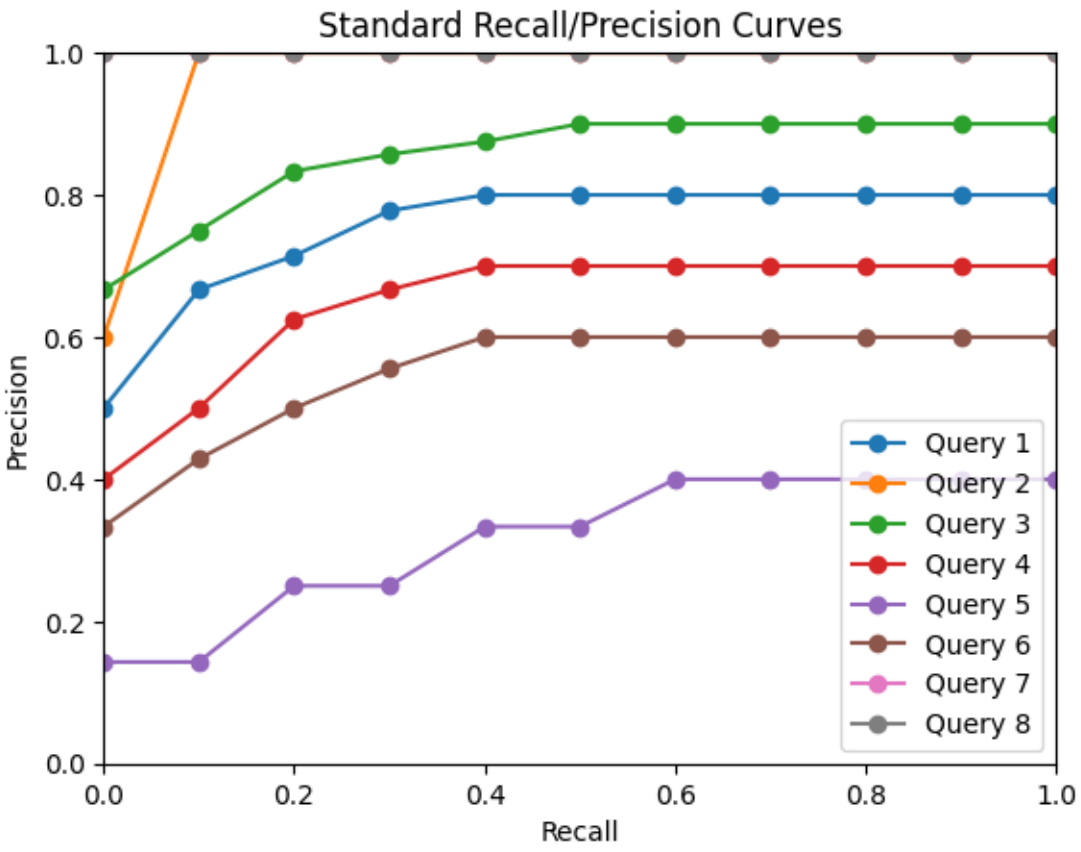
0.6666666666666666, 0.7]
Recall: [0.0, 0.0, 0.0, 0.07142857142857142, 0.14285714285714285,
0.21428571428571427, 0.2857142857142857, 0.35714285714285715,
0.42857142857142855, 0.5]
F1 Score [0, 0, 0, 0.11111111111111112, 0.21052631578947364, 0.3,
0.38095238095238093, 0.45454545454545453, 0.5217391304347826,
0.5833333333333334]

Query:5
Top n document IDs: [265, 45548, 8110, 4375, 41889, 38305, 13373, 17607, 41459,
33571]
Actual relevant ids:[13373, 17607, 41459, 54748, 33571]
Relevant [0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
precision: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.14285714285714285, 0.25,
0.3333333333333333, 0.4]
Recall: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 0.4, 0.6, 0.8]
F1 Score [0, 0, 0, 0, 0, 0, 0.16666666666666666, 0.3076923076923077,
0.42857142857142855, 0.5333333333333333]

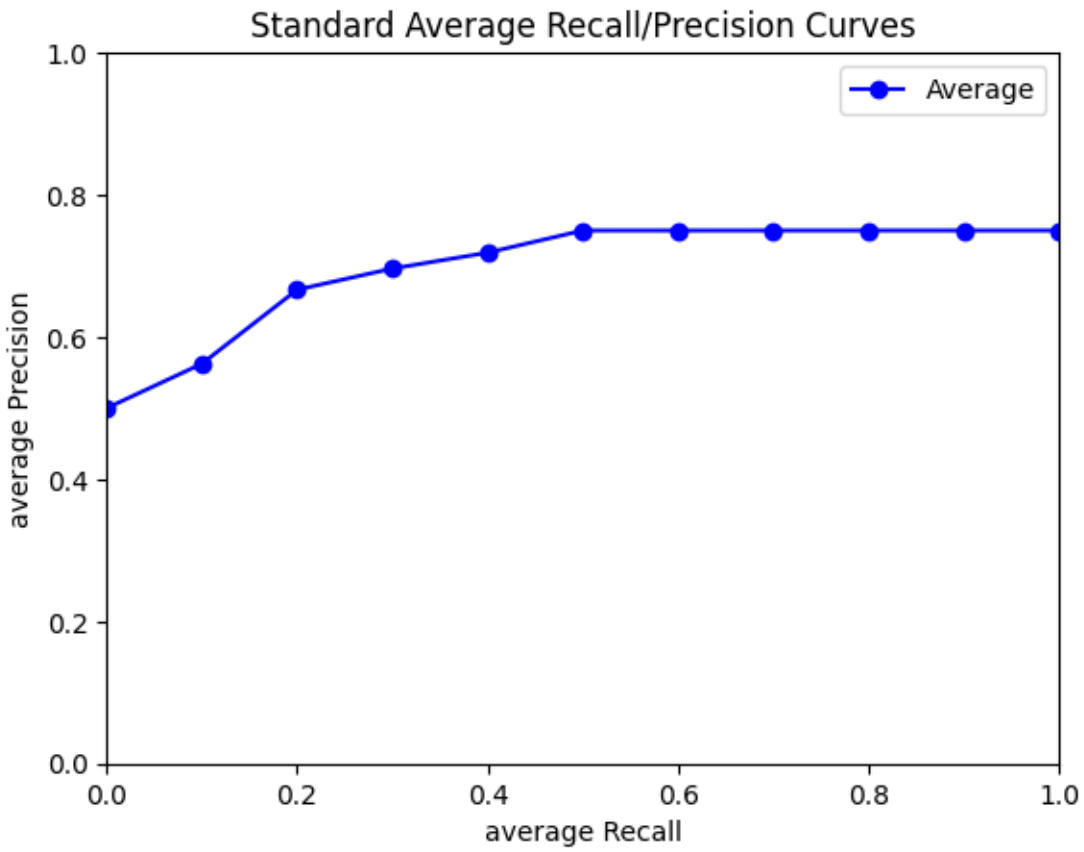
Query:6
Top n document IDs: [3865, 42077, 10612, 17166, 46500, 27474, 41319, 39932,
50197, 52375]
Actual relevant ids:[45860, 46500, 27474, 43945, 52837, 12358, 41319, 39932,
45146, 50197, 8341, 52375]
Relevant [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]
precision: [0.0, 0.0, 0.0, 0.0, 0.2, 0.3333333333333333, 0.42857142857142855,
0.5, 0.5555555555555556, 0.6]
Recall: [0.0, 0.0, 0.0, 0.0, 0.08333333333333333, 0.16666666666666666, 0.25,
0.3333333333333333, 0.4166666666666667, 0.5]
F1 Score [0, 0, 0, 0, 0.11764705882352941, 0.2222222222222222,
0.3157894736842105, 0.4, 0.4761904761904762, 0.5454545454545454]

Query:7
Top n document IDs: [209, 216, 11356, 38637, 7110, 6649, 51356, 44358, 36165,
943]
Actual relevant ids:[209, 28542, 216, 47345, 11356, 33632, 38637, 7110, 6649,
51356, 44358, 36165, 943, 37864]
Relevant [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
precision: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Recall: [0.07142857142857142, 0.14285714285714285, 0.21428571428571427,
0.2857142857142857, 0.35714285714285715, 0.42857142857142855, 0.5,
0.5714285714285714, 0.6428571428571429, 0.7142857142857143]

```
F1 Score [0.13333333333333333, 0.25, 0.35294117647058826, 0.4444444444444445,
0.5263157894736842, 0.6, 0.6666666666666666, 0.7272727272727273,
0.782608695652174, 0.8333333333333333]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Query:8
Top n document IDs: [642, 10642, 37794, 45518, 735, 10037, 41872, 53409, 56865,
44490]
Actual relevant ids:[642, 10642, 37794, 45518, 3494, 735, 10037, 41872, 28542,
53409, 56865, 44489, 44490]
Relevant [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
precision: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Recall: [0.07692307692307693, 0.15384615384615385, 0.23076923076923078,
0.3076923076923077, 0.38461538461538464, 0.46153846153846156,
0.5384615384615384, 0.6153846153846154, 0.6923076923076923, 0.7692307692307693]
F1 Score [0.14285714285714288, 0.2666666666666667, 0.375, 0.47058823529411764,
0.5555555555555556, 0.631578947368421, 0.7000000000000001, 0.761904761904762,
0.8181818181818181, 0.8695652173913044]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------

Average Recall, average Precision, average F1-measure:
average Recall@1~10:  [0.04 0.09 0.14 0.21 0.28 0.36 0.44 0.52 0.6  0.68]
average Precision@1~10:  [0.38 0.44 0.5  0.56 0.62 0.67 0.7  0.72 0.74 0.75]
average F1measure@1~10:  [0.07 0.14 0.22 0.29 0.38 0.45 0.52 0.58 0.64 0.69]

Standard Deviation of Recall, Precision, F1-measure:
Std Dev of Recall@1~10:  [0.06 0.11 0.16 0.21 0.24 0.28 0.24 0.21 0.18 0.17]
Std Dev of Precision@1~10:  [0.48 0.46 0.44 0.41 0.37 0.34 0.28 0.24 0.21 0.2 ]
Std Dev of F1measure@1~10:  [0.1  0.17 0.22 0.26 0.27 0.28 0.23 0.18 0.15 0.12]
```




```
<ipython-input-16-183a685fdef5>:127: UserWarning: color is redundantly defined
by the 'color' keyword argument and the fmt string "-bo" (-> color='b'). The
keyword argument will take precedence.
  plt.plot(x_axis, y_axis, '-bo', color="blue", label="Average")
```



4 Task 3 (NIR)

4.1 3a - Using DistilBert

Using the tranformer DistilBert, information retrieval can be acheived. This method begins by tokenising the text and encoding into numerical representations. This are then fed into the model which can capture the semantic meaning of the documents and queries. The created embeddings can then be used. For this task, the queries and data were passed throught the model to create embeddings for both query and the data. Similarly to task 2, cosine similarity was used to determine the most similar documents. Likewise, precision, recall and f1 scores were calculated to determine the performance of the model

[20]:
```
!pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-
packages (4.38.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from transformers) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.19.3 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.20.3)
```

```
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from transformers) (1.25.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (24.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (2023.12.25)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from transformers) (2.31.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.15.2)
Requirement already satisfied: safetensors>=0.4.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.4.2)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-
packages (from transformers) (4.66.2)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.19.3->transformers) (2023.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.19.3->transformers) (4.10.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->transformers) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (2024.2.2)
```

[21]:
```
import torch
import transformers as ppb
import warnings
```

[22]:
```
# For DistilBERT:
model_class, tokenizer_class, pretrained_weights = (ppb.DistilBertModel, ppb.
  ↪DistilBertTokenizer, 'distilbert-base-uncased')

# Load pretrained model/tokenizer
distilTokenizer = tokenizer_class.from_pretrained(pretrained_weights)
distilModel = model_class.from_pretrained(pretrained_weights)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
```

```
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
tokenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]
model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]
```

[23]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import torch
import transformers as ppb
import warnings
warnings.filterwarnings('ignore')

def embed_data(data):
  with torch.no_grad():
      # Tokenization
      tokenized = data.apply((lambda x: distilTokenizer.encode(x,␣
 ↪add_special_tokens=True)))

      # padding
      max_len = 0
      q = 0
      for i in tokenized.values:

          # BERT only accept maximum 512 values
          if len(i) > 512:
              temp = tokenized.values[q]
              tokenized.values[q] = temp[:512]
              i = tokenized.values[q]
              print('too much tokenized.values for BERT, only 512 are taken')

          # print(len(i))
          if len(i) > max_len:
              max_len = len(i)
          q += 1
```

```
    padded = np.array([i + [0]*(max_len-len(i)) for i in tokenized.values])
    np.array(padded).shape

    # masking
    attention_mask = np.where(padded != 0, 1, 0)
    attention_mask.shape

    # run the model
    input_ids = torch.tensor(padded)
    attention_mask = torch.tensor(attention_mask)


    last_hidden_states = distilModel(input_ids, attention_mask=attention_mask)

    train_features = last_hidden_states[0][:,0,:].numpy()

    return train_features
```

[24]:
```
# embed data
train_docs = pd.Series(data)
embeded_data = embed_data(train_docs)
```

[25]:
```
# get queries
file2 = '/content/drive/MyDrive/Colab Notebooks/COP509cw/Datasets/
 ↪JewelleryReviewsQueryRelevantID.csv'
querydf = pd.read_csv(file2)
queries = get_queries(querydf)

# mapping from id to vectorized document
id_to_vectorized = dict(zip(ids, embeded_data))

j=1
n = 10
AllRecall = []
AllPrecision = []
AllF1measure = []

for query, query_relevant_ids in zip(queries, relevant_ids):

    query = clean_data(query)
    query_line = ' '.join(query)
    query_docs = pd.Series(query_line)
    embeded_query = embed_data(query_docs)
    similarities = cosine_similarity(embeded_data, embeded_query)

    indexes = np.argsort(similarities.flat)[-n:]
```

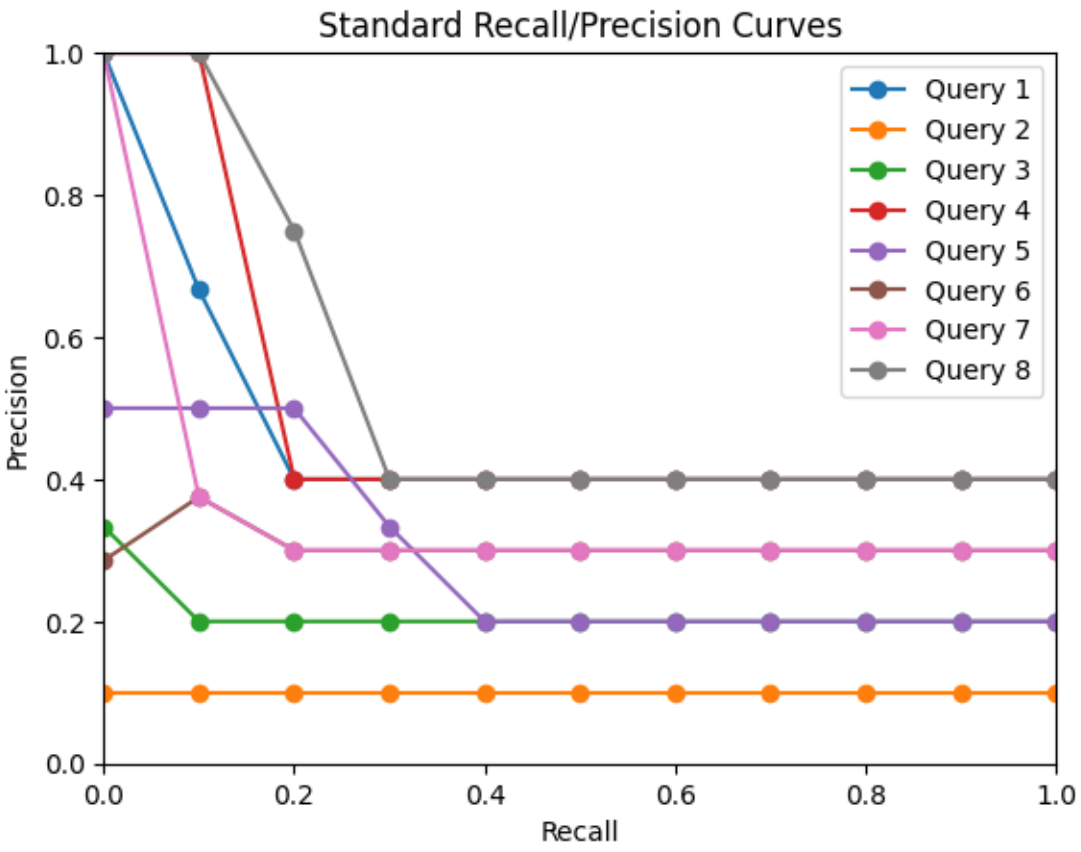
```
    # Retrieve the top n vectorized documents
    top_n_vectorized = data_vectorized[indexes]

    # Find the indices of the top n similarity values
    top_n_indices = np.argsort(similarities.flat)[-n:][::-1]

    # Map the indices back to the corresponding IDs
    top_n_ids_nir = [ids[idx] for idx in top_n_indices]

    # Mark the relevant index
    re_mark = [1 if top_n_id in query_relevant_ids else 0 for top_n_id in␣
↪top_n_ids_nir]

    print("Query:{}".format(j))

    print("Top n document IDs: " + str(top_n_ids_lsi))
    print("Actual relevant ids:" + str(query_relevant_ids))

    print("Relevant", re_mark)

    precision = calculate_precision(re_mark)
    print("precision:", precision)

    recall = calculate_recall(re_mark, query_relevant_ids)
    print("Recall:", recall)

    F1 = calculate_f1(precision, recall)
    print("F1 Score", F1)

    print("-" * 200)

    # save
    AllRecall.append(recall)
    AllPrecision.append(precision)
    AllF1measure.append(F1)

    # plot R/P curve for each query
    x_axis = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
    y_axis = compute_RP_yaxis(Precision=precision, Recall=recall)
    plt.plot(x_axis, y_axis, '-o', label="Query %d" % (j))

    j += 1

# compute average Recall, average Precision, average F1-measure
AllRecall = np.array(AllRecall)
AllPrecision = np.array(AllPrecision)
AllF1measure = np.array(AllF1measure)
```

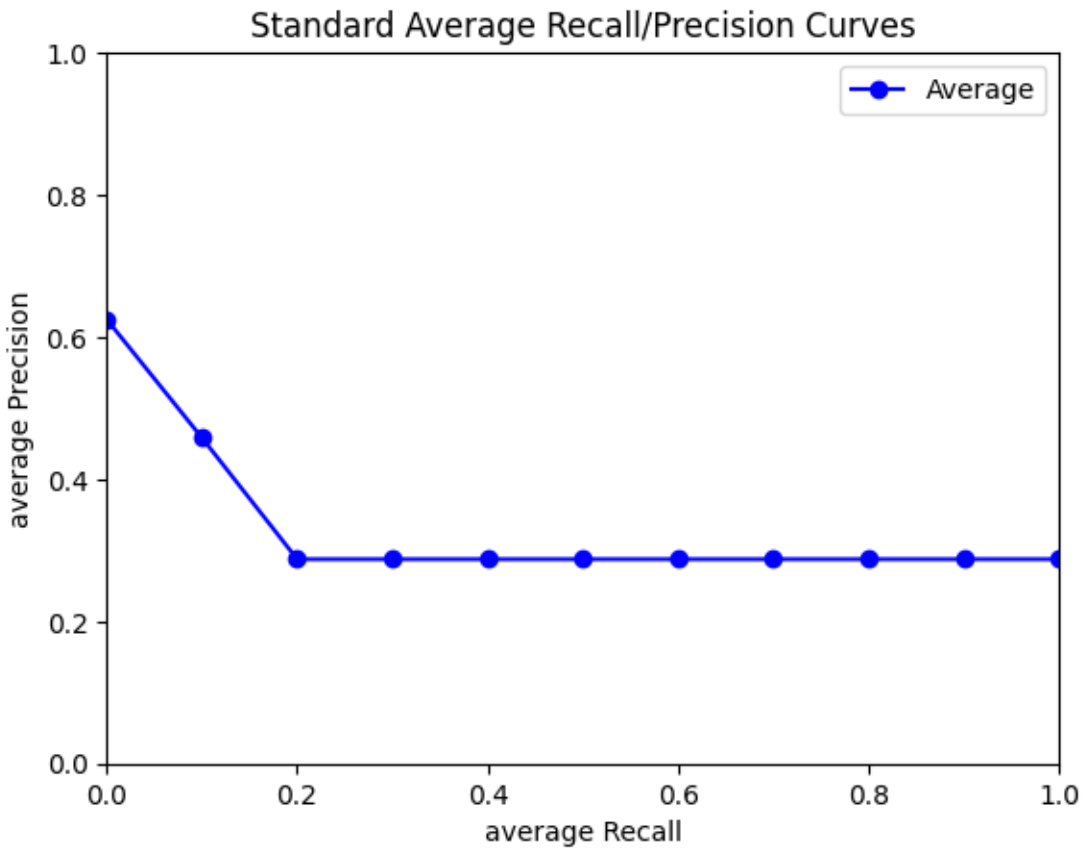
```
# compute average Recall, average Precision, average F1-measure
NIRAveRecall = np.mean(AllRecall, axis=0)
NIRAvePrecision = np.mean(AllPrecision, axis=0)
NIRAveF1measure = np.mean(AllF1measure, axis=0)

# Calculate standard deviation
NIRStdRecall = np.std(AllRecall, axis=0)
NIRStdPrecision = np.std(AllPrecision, axis=0)
NIRStdF1measure = np.std(AllF1measure, axis=0)

print("\nAverage Recall, average Precision, average F1-measure: ")
print("average Recall@1~10: ", np.around(NIRAveRecall[:10],2))
print("average Precision@1~10: ", np.around(NIRAvePrecision[:10],2))
print("average F1measure@1~10: ", np.around(NIRAveF1measure[:10],2))

print("\nStandard Deviation of Recall, Precision, F1-measure: ")
print("Std Dev of Recall@1~10: ", np.around(NIRStdRecall[:10], 2))
print("Std Dev of Precision@1~10: ", np.around(NIRStdPrecision[:10], 2))
print("Std Dev of F1measure@1~10: ", np.around(NIRStdF1measure[:10], 2))

# Display the overall plot after all queries

plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Standard Recall/Precision Curves')
plt.legend()
plt.savefig("nir_pr.png")
plt.show()


# plot average R/P curve

x_axis = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
y_axis = compute_RP_yaxis(Precision=NIRAvePrecision, Recall=NIRAveRecall)
plt.plot(x_axis, y_axis, '-bo', color="blue", label="Average")
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel('average Recall')
plt.ylabel('average Precision')
plt.title('Standard Average Recall/Precision Curves')
plt.legend()
plt.savefig("nir_pr_average.png")
plt.show()
```

```
Query:1
Top n document IDs: [642, 10642, 37794, 45518, 735, 10037, 41872, 53409, 56865,
44490]
Actual relevant ids:[36164, 58481, 26246, 2033, 48779, 34523, 9726, 56494,
49525, 45278, 35694, 41876, 17309, 11135, 17273, 11247]
Relevant [1, 0, 1, 0, 0, 1, 0, 0, 1, 0]
precision: [1.0, 0.5, 0.6666666666666666, 0.5, 0.4, 0.5, 0.42857142857142855,
0.375, 0.4444444444444444, 0.4]
Recall: [0.0625, 0.0625, 0.125, 0.125, 0.125, 0.1875, 0.1875, 0.1875, 0.25,
0.25]
F1 Score [0.11764705882352941, 0.1111111111111111, 0.21052631578947367, 0.2,
0.19047619047619047, 0.2727272727272727, 0.26086956521739124, 0.25, 0.32,
0.3076923076923077]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Query:2
Top n document IDs: [642, 10642, 37794, 45518, 735, 10037, 41872, 53409, 56865,
44490]
Actual relevant ids:[57123, 25299, 55017, 7432, 2114, 40871]
Relevant [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
precision: [1.0, 0.5, 0.3333333333333333, 0.25, 0.2, 0.16666666666666666,
0.14285714285714285, 0.125, 0.1111111111111111, 0.1]
Recall: [0.16666666666666666, 0.16666666666666666, 0.16666666666666666,
0.16666666666666666, 0.16666666666666666, 0.16666666666666666,
0.16666666666666666, 0.16666666666666666, 0.16666666666666666,
0.16666666666666666]
F1 Score [0.2857142857142857, 0.25, 0.2222222222222222, 0.2, 0.1818181818181818,
0.16666666666666666, 0.15384615384615383, 0.14285714285714288,
0.13333333333333333, 0.125]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Query:3
Top n document IDs: [642, 10642, 37794, 45518, 735, 10037, 41872, 53409, 56865,
44490]
Actual relevant ids:[33251, 17304, 50019, 27679, 6158, 22408, 29722, 36677,
2780, 17944, 19944, 31657, 52867, 49216]
Relevant [0, 0, 1, 0, 0, 0, 1, 0, 0, 0]
precision: [0.0, 0.0, 0.3333333333333333, 0.25, 0.2, 0.16666666666666666,
0.2857142857142857, 0.25, 0.2222222222222222, 0.2]
Recall: [0.0, 0.0, 0.07142857142857142, 0.07142857142857142,
0.07142857142857142, 0.07142857142857142, 0.14285714285714285,
0.14285714285714285, 0.14285714285714285, 0.14285714285714285]
F1 Score [0, 0, 0.11764705882352941, 0.1111111111111112, 0.10526315789473682,
0.1, 0.19047619047619047, 0.18181818181818182, 0.17391304347826086,
0.16666666666666666]
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
--------------------------------------
Query:4
Top n document IDs: [642, 10642, 37794, 45518, 735, 10037, 41872, 53409, 56865,
44490]
Actual relevant ids:[40373, 28648, 37486, 30640, 2131, 19852, 2134, 36585,
26535, 51474, 21070, 56330, 53660, 44126]
Relevant [1, 1, 1, 0, 0, 1, 0, 0, 0, 0]
precision: [1.0, 1.0, 1.0, 0.75, 0.6, 0.6666666666666666, 0.5714285714285714,
0.5, 0.4444444444444444, 0.4]
Recall: [0.07142857142857142, 0.14285714285714285, 0.21428571428571427,
0.21428571428571427, 0.21428571428571427, 0.2857142857142857,
0.2857142857142857, 0.2857142857142857, 0.2857142857142857, 0.2857142857142857]
F1 Score [0.13333333333333333, 0.25, 0.35294117647058826, 0.3333333333333333,
0.3157894736842105, 0.4, 0.38095238095238093, 0.36363636363636365,
0.34782608695652173, 0.3333333333333333]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------
Query:5
Top n document IDs: [642, 10642, 37794, 45518, 735, 10037, 41872, 53409, 56865,
44490]
Actual relevant ids:[13373, 17607, 41459, 54748, 33571]
Relevant [0, 1, 0, 0, 0, 1, 0, 0, 0, 0]
precision: [0.0, 0.5, 0.3333333333333333, 0.25, 0.2, 0.3333333333333333,
0.2857142857142857, 0.25, 0.2222222222222222, 0.2]
Recall: [0.0, 0.2, 0.2, 0.2, 0.2, 0.4, 0.4, 0.4, 0.4, 0.4]
F1 Score [0, 0.28571428571428575, 0.25, 0.22222222222222224,
0.20000000000000004, 0.3636363636363636, 0.3333333333333333, 0.3076923076923077,
0.2857142857142857, 0.26666666666666666]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------
Query:6
Top n document IDs: [642, 10642, 37794, 45518, 735, 10037, 41872, 53409, 56865,
44490]
Actual relevant ids:[45860, 46500, 27474, 43945, 52837, 12358, 41319, 39932,
45146, 50197, 8341, 52375]
Relevant [0, 0, 0, 1, 0, 0, 1, 1, 0, 0]
precision: [0.0, 0.0, 0.0, 0.25, 0.2, 0.16666666666666666, 0.2857142857142857,
0.375, 0.3333333333333333, 0.3]
Recall: [0.0, 0.0, 0.0, 0.08333333333333333, 0.08333333333333333,
0.08333333333333333, 0.16666666666666666, 0.25, 0.25, 0.25]
F1 Score [0, 0, 0, 0.125, 0.11764705882352941, 0.1111111111111111,
0.2105263157894737, 0.3, 0.28571428571428575, 0.2727272727272727]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------
```

```
Query:7
Top n document IDs: [642, 10642, 37794, 45518, 735, 10037, 41872, 53409, 56865,
44490]
Actual relevant ids:[209, 28542, 216, 47345, 11356, 33632, 38637, 7110, 6649,
51356, 44358, 36165, 943, 37864]
Relevant [1, 0, 0, 0, 0, 1, 0, 1, 0, 0]
precision: [1.0, 0.5, 0.3333333333333333, 0.25, 0.2, 0.3333333333333333,
0.2857142857142857, 0.375, 0.3333333333333333, 0.3]
Recall: [0.07142857142857142, 0.07142857142857142, 0.07142857142857142,
0.07142857142857142, 0.07142857142857142, 0.14285714285714285,
0.14285714285714285, 0.21428571428571427, 0.21428571428571427,
0.21428571428571427]
F1 Score [0.13333333333333333, 0.125, 0.11764705882352941, 0.11111111111111112,
0.10526315789473682, 0.2, 0.19047619047619047, 0.2727272727272727,
0.2608695652173913, 0.25]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Query:8
Top n document IDs: [642, 10642, 37794, 45518, 735, 10037, 41872, 53409, 56865,
44490]
Actual relevant ids:[642, 10642, 37794, 45518, 3494, 735, 10037, 41872, 28542,
53409, 56865, 44489, 44490]
Relevant [1, 1, 0, 1, 0, 0, 0, 1, 0, 0]
precision: [1.0, 1.0, 0.6666666666666666, 0.75, 0.6, 0.5, 0.42857142857142855,
0.5, 0.4444444444444444, 0.4]
Recall: [0.07692307692307693, 0.15384615384615385, 0.15384615384615385,
0.23076923076923078, 0.23076923076923078, 0.23076923076923078,
0.23076923076923078, 0.3076923076923077, 0.3076923076923077, 0.3076923076923077]
F1 Score [0.14285714285714288, 0.2666666666666667, 0.25, 0.3529411764705882,
0.33333333333333337, 0.3157894736842105, 0.3, 0.380952380952381,
0.3636363636363637, 0.34782608695652173]
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------

Average Recall, average Precision, average F1-measure:
average Recall@1~10:  [0.06 0.1  0.13 0.15 0.15 0.2  0.22 0.24 0.25 0.25]
average Precision@1~10:  [0.62 0.5  0.46 0.41 0.32 0.35 0.34 0.34 0.32 0.29]
average F1measure@1~10:  [0.1  0.16 0.19 0.21 0.19 0.24 0.25 0.27 0.27 0.26]

Standard Deviation of Recall, Precision, F1-measure:
Std Dev of Recall@1~10:  [0.05 0.07 0.07 0.06 0.06 0.1  0.08 0.08 0.08 0.08]
Std Dev of Precision@1~10:  [0.48 0.35 0.29 0.21 0.17 0.18 0.12 0.12 0.12 0.11]
Std Dev of F1measure@1~10:  [0.09 0.11 0.1  0.09 0.08 0.11 0.07 0.08 0.08 0.07]
```


Standard Recall/Precision Curves
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Query 1
Query 2
Query 3
Query 4
Query 5
Query 6
Query 7
Query 8



4.2 3B Results comparison

4.2.1 Plots

[26]:
```
from IPython.display import Image, display
import matplotlib.image as mpimg

# Specify the filenames for your images
image_filenames = ["/content/best_model_pr.png", "/content/nir_pr.png", "/
 ↪content/best_model_pr_average.png", "/content/nir_pr_average.png"]

# Create a 2x2 grid of subplots
fig, axes = plt.subplots(2, 2, figsize=(10, 10))

# Titles for the images
image_titles = ["Figure 1: Best LSI model", "Figure 2: NIR model", "Figure 3:␣
 ↪Best LSI model", "Figure 4: NIR model"]

# Iterate through the images and display them in the grid
for i, ax in enumerate(axes.flat):
```

```
    img = mpimg.imread(image_filenames[i])
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(image_titles[i])  # Set the title based on the index

# Adjust layout for better spacing
plt.tight_layout()

# Show the plot
plt.show()
```

Figure 1: Best LSI model

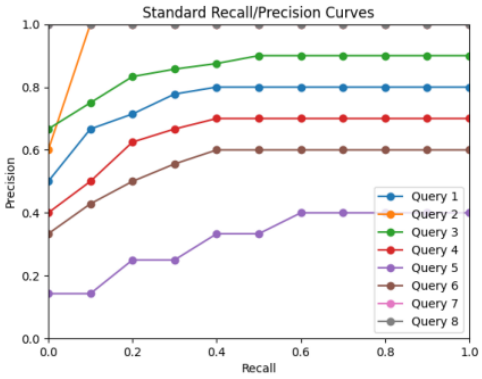


Figure 2: NIR model

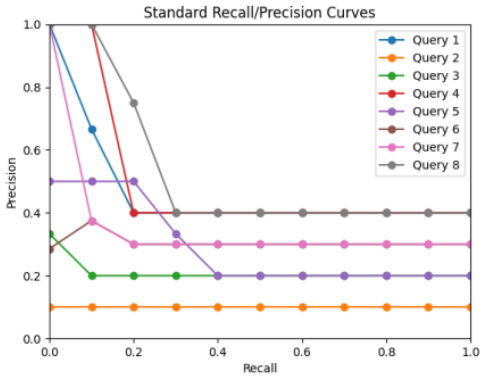


Figure 3: Best LSI model

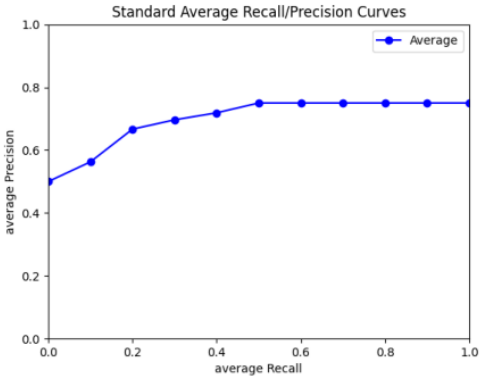
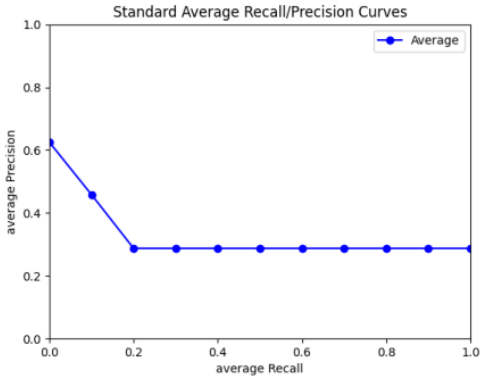


Figure 4: NIR model



4.2.2 Table of results

[27]:
```
# Define the data
LSIAverage = [BestAveRecall[9], BestAvePrecision[9], BestAveF1measure[9]]
NIRAverage = [NIRAveRecall[9], NIRAvePrecision[9], NIRAveF1measure[9]]
LSISTD = [BestStdRecall[9], BestStdPrecision[9], BestStdF1measure[9]]
```

```
NIRSTD = [NIRStdRecall[9], NIRStdPrecision[9], NIRStdF1measure[9]]

# Create a dictionary with the data
results = {
    'Best LSI Average': LSIAverage,
    'NIR Average': NIRAverage,
    'LSI Standard deviation': LSISTD,
    'NIR standard deviation': NIRSTD
}


resultsdf = pd.DataFrame(results, index=['Recall', 'Precision', 'F1 Measure'])

resultsdf
```

[27]:

	Best LSI Average	NIR Average	LSI Standard deviation	\
Recall	0.678297	0.252152	0.167516	
Precision	0.750000	0.287500	0.200000	
F1 Measure	0.685051	0.258739	0.123580	

	NIR standard deviation
Recall	0.076458
Precision	0.105327
F1 Measure	0.072969

4.2.3 Results discussion

As shown above, Figure 1 displays the recall - precision graph for each query for the best tuned LSI model. In comparison, Figure 2 displays the recall - precision graph for the NIR model. As observed, the LSI model performs much better for each query with an increase in precision though the documents however, the NIR model struggles to perform well. In addition, the overall averages are calculated for the queries for each document as displayed in figure 3 and 4. In figure 3, this shows that the LSI model performs really well and gradually decreases in precision throughout the docuemnts. This is to be expected as the number of relevant documents might be less than the number of retrieved documents (in this case - 10). However, as displayed in figure 4, the NIR on average can retrieve a few relavent documents for the first few documents and then stops accuratly retrieving relavent documents almost instantly. In the table above displays the averages and standard deviation of the F1 score calculated after the 10th retrieved document. The results further indicate that LSI model performs much better than the NIR model.

4.3 3C - Interactive UI using best model

These code blocks use IPy widgets and the LSI model (with the best tuned parameters) to allow a user to type in a query and the number of relative documents they would like to retrieve. The ids and assiociated reviews (as raw text) is displayed

[28]:
```
all_reviews = []

def retrieve_top_docs(_):

    clear_output(wait=True)
    display(allw)
    query = query_field.value
    n = n_doc_field.value

    # Check if query or n is empty
    if not query or not n:
        print("Please enter both a query and a value for the number of␣
↪documents.")
        return

    try:
        n = int(n)
        if n <= 0:
            print("Please enter a valid positive value for the number of␣
↪documents.")
            return
    except ValueError:
        print("Please enter a valid numeric value for the number of documents.")
        return

    # SVD on data
    data_transformed = vectorize_data(data, vocab, best_norm)
    trunc_SVD_model = TruncatedSVD(n_components=best_n_comp)
    data_vectorized = trunc_SVD_model.fit_transform(data_transformed)

    # mapping from id to vectorized document
    id_to_vectorized = dict(zip(ids, data_vectorized))

    # Find similarities for each query
    processed_query = process_query(query, vocab, best_norm)

    transformed_query = trunc_SVD_model.transform(processed_query)
    similarities = cosine_similarity(data_vectorized, transformed_query)
    indexes = np.argsort(similarities.flat)[-n:]

    # Retrieve the top n vectorized documents
    top_n_vectorized = data_vectorized[indexes]

    # Map the vectorized documents back to their corresponding IDs
    top_n_ids = [id for id, vectorized in id_to_vectorized.items() if␣
↪vectorized in top_n_vectorized]
```

```
    for id in top_n_ids:

        # get the raw review for printing
        raw_review = id_to_raw_data.get(id, None)
        # get cleaned review for future use
        cleaned_review = id_to_clean_data.get(id, None)
        all_reviews.append(cleaned_review)

        print(id, raw_review)
        print("-" * 200)

    return
```

[29]:
```
# Search button
search_button = widgets.Button(description="Search")
search_button.on_click(retrieve_top_docs)

# query field
query_field = widgets.Text(description="Enter Query:")
n_doc_field = widgets.Text(description="Enter the number of related documents␣
 ↪you wish to retrieve:")

#H layout
fields = widgets.HBox([query_field, n_doc_field])

output = widgets.Output()

allw = widgets.VBox([fields, search_button, output])
```

[30]:
```
# display widgets
display(allw)
```

```
VBox(children=(HBox(children=(Text(value='This is really great',␣
 ↪description='Enter Query:'), Text(value='50',…
```

```
43515 This ring is a beautiful ring but the first shipment of this ring had
scratches on the diamond and was missing 2 diamonds on the shoulder of the ring.
I returned the ring for a replacement of the same ring, the second ring was even
worse. Clearly visible white scratch right in the middle of a black diamond does
not look that great. And this time 1 diamond missing on the shoulder of the
ring. I returned it for a refund. Thanks alot Amazon.
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------
28250 I wanted to know if this ring is like 2 rings in one, because this ring is
beyond gorgeous, I just love it.
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
```

36727 I only had it a week before one of the jewels fell out of the wing. The metal is already tarnishing too. Spend your money on a higher quality item.

11087 its what i wanted :) but its not my favorite piercing of mine but i have to wear the bioplast cuz i break out with certain metals
---

48216 I got this ring for my birthday and I love it, I cannot imagine a woman not adoring this ring.

50640 This ring is awesome! I wear the ring on my index finger and it fits great! Comfortable ring.

21185 I bought this ring as a Valentine's gift but couldn't give it since the hands holding the heart symbol would reverse whenever the ring spins (there are 4 of them on this ring, alternating the facing direction). It's a cute ring so I just kept it for myself.

44591 my husband loves it only thing is you cant have this ring resized due to the way the ring is made
---

12483 They definitely help lessen your appitite, however my ears were sore after wearing for about 3 hours and the next few days I tried to wear them off and on and to increase the wearing time. If you have a good pain tolerance you may not notice any discomfort, as for me my ears lobes were swollen and I had to stop wearing them for 4 days.

37896 my only wish on this ring is- I wish the cut potrion went all the way around the ring. Other than that a great very comfortable ring.

45856 I could see the quality work in this rosary. I was pleased when I opened the container and I could immediately smell olive wood. There was only one thing

buyers need to be aware of when buying this item that wasn't mentioned in the description. The Holy Land Sand is present in a compartment on the rosary. In my opinion, this adds to the quality of the rosary so I'm not sure why the seller didn't describe this fact. I'll be looking for more items from this seller.

33746 i love this ring my only complaint is that the metal is so soft that it bent to the point that i couldnt wear it anymore. i still wear it on a necklace and love it on occasion i can squish it onto my fingure but it took less than two months for it to be almost impossible for me to wear.

3865 What sparkle. It is so pretty and dainty. Just what I was looking for.

34483 i love the ring it shines beautiful its looks so real anyone who like to buy one this is the ring the stones are a good size my friends love the ring it could pass for a real diamond i love the ring its worth the money its better then the ring i bought from avon it was silver and had cubic z no shine to the ring but this ring i bought with the 3 stones its worth the price anne marie
---

40749 got this for my gf not as an enguagement ring and much to my supprize it was two rings which fit together to make 1. it looked like just one ring in the picture. i must say it is a beautiful ring which always catches peoples attention
---

6522 I love the ring and suggest every girl should have this ring in their jewelry collection.

45289 I bought this for my 17 year old daughter who had seen a small heart shaped ring at the Coach store and liked the style. This one was much cheaper and ended up being so much prettier! It's really an eye catching ring, doesn't look cheap and sparkles beautifully. I am actually ordering her a second ring since she will be leaving for college next month and is worried about losing it. I think she'll like knowing there is a spare ring at home just in case and for the price it's not a big deal. She wears this ring on her right ring finger instead of her high school ring because she loves it that much. I would suggest this ring for any fan of hearts and a little bit of bling!

14499 This ring is exactly what I wanted, I actually bought another and it wasn't quite as solidly made as this ring is. Thanks for the great ring!

48772 Great ring! I tried a different one first but the individual rings were so thin that they snapped within a few weeks. The rings on this one are thicker, making it a bolder and more durable ring. Of course it costs a bit more but you can't go wrong paying for better quality!

51907 I was not completely sure about buying jewlery over the Internet. For this type of gift I like to see and handle the item to decide. I was pressed for time and I did like the pictures of the item so I decided to take a chance. I'm glad I did as I was very impressed by the quality and appearance when the item arrived. The price was low compared to the quality and my wife and I were very pleased with this jewlery item.

22058 Item was great quality and came promptly. I'm very happy with it and recommend it unreservedly.

36164 I got the ring as a promise ring for my girlfriend for Christmas and she loved it. Definitely a great value.

58481 my wife loves the ring, it was a great gift. extremelly cheap and high quality.

26246 This was a birthday gift for my 16 YO niece. She loves the ring and was very happy to have received it.

2033 I love my birthstone and I wanted a piece of jewelry that symbolized the simple purity of the Blue Topaz. This ring did that for me. As a gift to myself for my birthday this year, it was definitely a great gift and a welcomed addition to my collection.

48779 i got this ring as a gift from my boyfriend and i love it. the only thing is that if the rings are not position correctly it pinches the skin.

34523 This ring has such a good sparkle and it looks like a ring that should cost 10x the amount. Makes a great gift for someone on a budget. My girlfriend loves it.

9726 A great gift to your loved one and an ever better seller. The seller deals with you in the most professional way and the security measures are superb.

56494 I bought this ring for my husband and he loved it. I received it when they said I would and it is a great ring
---

49525 this product made for a great gift and great memorize for my love and me. It something we will always have. a helping gift from the heart that always shows you care.

45278 I love this ring fits just right and I showed my daughter the ring and she loved it as well. Great for everyday wear and the price was great..

35694 My neice loves her birth stone so I got it for her for a Christmas Gift.I also love it also. great
---

41876 I bought this as a gift for a friends birthday and she loved it. It's a beautifull ring.

17309 I always love Willow Tree. they make great gifts for great people in your life. I have quite a collection, and I hope to continue to build it
---

11135 I've owned several Claddagh rings over the years and lost my last one. I received this one as a gift recently and I truly love it. It's sturdy and the design is crisp. I can easily recommend this one as a lovely ring.

17273 My mother loved this and was a great birthday gift. These look even better in person and go great with anything.

11247 I have always wanted a claddaugh ring.This price was great.I love it
---

40373 I bought this to wear with my alice in wonderland costume for halloween, but I've been wearing it ever since I received it and have actually gotten a lot of compliments on it :)

28648 The days I do not wear the blue one I wear this one. I really enjoy wearing something Celtic and pretty.

2131 I wanted a classy piece to wear on my right hand for work when I'm wearing Gold. I found that I will end up wearing this outside of work. Very classy looking
---

2134 ery suitable for wearing for fashionable occasions. very dressy
---

36585 I am looking forward to wearing them as they sparkle and catch every eye at my son's wedding on June 30

56330 great way to support the local pro sports team without wearing an oversized jersey or a hat to mess up the hair
---

45860 This is one of the most beautiful rosarys I have seen. The smoothness and

```
color of the beads is so translucent looking that it almost looks like glass.
The workmanship is excellent and the details are beautiful. A truly beautiful
piece to own.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---------------------------------------
52837 I can imagine this sparkling around my girlfriends tanned toe in the sun.
Too bad its winter. But I will be looking forward to. It should be a pretty
sight.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---------------------------------------
12358 These look quite like their photograph. They are very colorful and you
know they are turtles. I've seen them elsewhere for quite a high price and these
are beautiful.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---------------------------------------
39932 This dainty heart looks absolutely beautiful on. It picks up the colors of
your clothing. It is an amazing price for such a beautiful pendant.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---------------------------------------
45146 You will not find a better deal for the money anywhere. This ring is so
beautiful. The craftsmanship is meticulous and extensive. Also, the mussel
shells are pinker in person than they look online. They are deep in color in
various shades of iridescent mauve.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---------------------------------------
8341 The smaller carnelian beads have a clear beautiful orange color. It would
be great if the elastic was the same color of the beads. Very beautiful.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---------------------------------------
44489 I just got these yesterday as a Christmas gift- so far they look just like
the picture and seem very nice.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---------------------------------------
```

5 Task 4 (Clustering)

5.1 4a - Bert topic

The following code blocks use Berttopic for topic modelling. The text data is first transformed using the pre-trained model and embedded to capture the semantic meaning. The model then undergoes dimensional reduction using SVD and then clustered using Kmean.

To run the following code in this section, make sure to run the code for section 3C and enter a query and number of retrieved documents. The following example uses 50 retrieved documents obtained from the model in task 3C for the query 'This is really great'.

[31]:
```
%%capture
!pip install bertopic
```

[32]:
```
# print to see the reviews entering the Bert topic model

#print(all_reviews)
```

[33]:
```
print(len(all_reviews))
```

```
50
```

[34]:
```
# Please select from (UMAP, SVD, PCA)
RedDiMethod = 'SVD'
```

[44]:
```
from sentence_transformers import SentenceTransformer
from umap import UMAP
from sklearn.decomposition import TruncatedSVD, PCA
from sklearn.cluster import KMeans
from hdbscan import HDBSCAN
from sklearn.feature_extraction.text import CountVectorizer
from bertopic.vectorizers import ClassTfidfTransformer
from bertopic import BERTopic
from sklearn.cluster import DBSCAN

# Step 1 - Extract embeddings
embedding_model = SentenceTransformer("all-MiniLM-L6-v2")

# Step 2 - Reduce dimensionality
if RedDiMethod == 'SVD':
   RD_model = TruncatedSVD(n_components=20)
elif RedDiMethod == 'PCA':
   RD_model = PCA(n_components='mle')
else:
   RD_model = UMAP(n_neighbors=15, n_components=5, min_dist=0.0,␣
 ↪metric='cosine')

n_clusters = 5
# Step 3 - Cluster reduced embeddings
Cluster_model = KMeans(n_clusters=n_clusters, random_state=42)


# Step 4 - Tokenize topics
vectorizer_model = CountVectorizer(stop_words="english")
```

```
# Step 5 - Create topic representation
ctfidf_model = ClassTfidfTransformer()

# All steps together
topic_model = BERTopic(
  embedding_model=embedding_model,    # Step 1 - Extract embeddings
  umap_model=RD_model,                # Step 2 - Reduce dimensionality
  hdbscan_model=Cluster_model,        # Step 3 - Cluster reduced embeddings
  vectorizer_model=vectorizer_model,  # Step 4 - Tokenize topics
  ctfidf_model=ctfidf_model,          # Step 5 - Extract topic words
  calculate_probabilities=True
)

topics, probs = topic_model.fit_transform(all_reviews)
```

Here shows the count and key words for each cluster

[45]:
```
freq = topic_model.get_topic_info()
freq.head(n_clusters)
```

[45]:
```
   Topic  Count                            Name  \
0      0     20          0_ring_love_great_bought
1      1     12         1_gift_item_great_quality
2      2      8              2_ring_wear_ear_cut
3      3      6  3_beautiful_looking_bead_color
4      4      4       4_wearing_classy_work_wear

                                      Representation  \
0  [ring, love, great, bought, diamond, like, loo…
1  [gift, item, great, quality, seller, jewelry, …
2  [ring, wear, ear, cut, couldn, wish, comfortab…
3  [beautiful, looking, bead, color, forward, pre…
4  [wearing, classy, work, wear, hat, pro, sport,…

                                 Representative_Docs
0  [ring beautiful ring first shipment ring scrat…
1  [love birthstone wanted piece jewelry symboliz…
2  [bought ring valentine gift couldn't give sinc…
3  [imagine sparkling around girlfriend tanned to…
4  [day wear blue one wear one really enjoy weari…
```

This shows the most frquent words for the first cluster

[46]:
```
topic_model.get_topic(0)
```

[46]:
```
[('ring', 0.18604069640430987),
 ('love', 0.07982530867265855),
 ('great', 0.05914236175572094),
 ('bought', 0.05352098136161623),
 ('diamond', 0.05136636650087591),
 ('like', 0.041441590691185),
 ('look', 0.041441590691185),
 ('loved', 0.03898129889712401),
 ('beautiful', 0.03778460972256646),
 ('price', 0.037207387001368566)]
```

Here shows the cluster distance on a graph. Note that this does not show when convert to pdf:

[47]:
```
topic_model.visualize_topics()
```

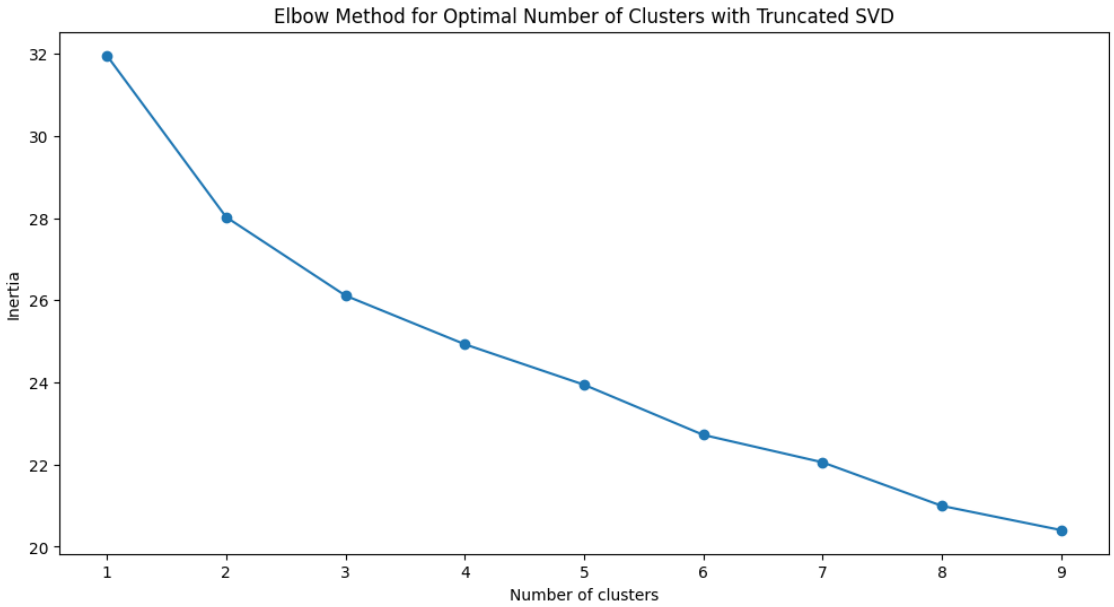
5.2 4B - Finding optimal number of clusters

5.2.1 Elbow Method

To find the optimal number of clusters, two methods were explored to find the best number clusters for this data. Firstly, the elbow method was used as shown below. The optimal number of clusters is where the rate of decrease of interia to clusters slows down i.e. the elbow.

[48]:
```
# source: https://www.analyticsvidhya.com/blog/2019/08/
 ↪comprehensive-guide-k-means-clustering/#:~:
 ↪text=One%20commonly%20used%20method%20to,distances%20begins%20to%20level%20off.
 ↪

from sentence_transformers import SentenceTransformer
from sklearn.cluster import KMeans
from sklearn.decomposition import TruncatedSVD
import pandas as pd
import matplotlib.pyplot as plt

# Step 1 - Extract embeddings
embedding_model = SentenceTransformer("all-MiniLM-L6-v2")
embeddings = embedding_model.encode(all_reviews)


# Fitting multiple k-means algorithms and storing the values in an empty list
SSE = []
for cluster in range(1, 10):
    kmeans = KMeans(n_clusters=cluster, init='k-means++')
    kmeans.fit(embeddings)  # Use the reduced embeddings for clustering
    SSE.append(kmeans.inertia_)

# Converting the results into a dataframe and plotting them
frame = pd.DataFrame({'Cluster': range(1, 10), 'SSE': SSE})
```

```
plt.figure(figsize=(12, 6))
plt.plot(frame['Cluster'], frame['SSE'], marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal Number of Clusters with Truncated SVD')
plt.show()
```



As shown above, the gradient slows down at 2 - 4 clusters, so between 2 - 4 clusters is optimal

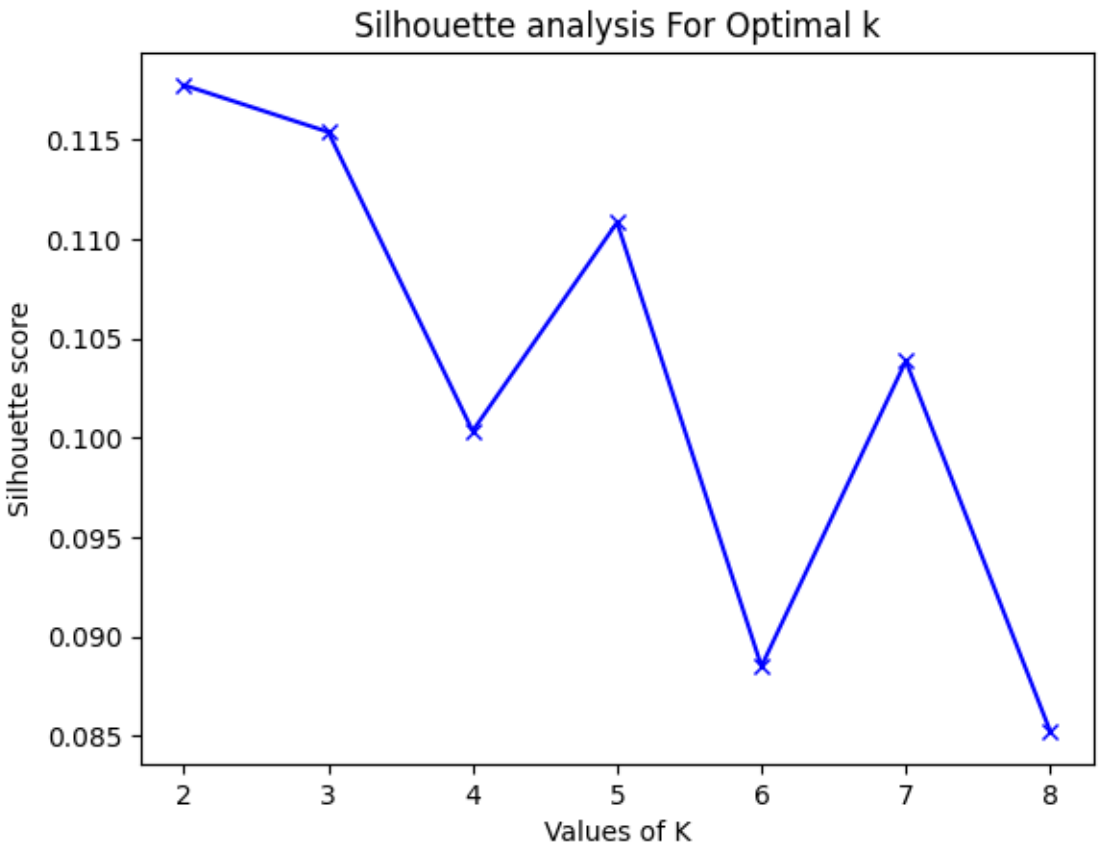
5.2.2 Silhouette Method

The silhouette method evaluates the quality of clusters based on how well-separated they are and how close the data points are to their own clusters compared to neighboring clusters. The silhouette score measures this cohesion and separation, with values ranging from -1 to 1, where higher values indicate better clustering. The point at which the score, is greatest, is the most optimal number of clusters

[55]:
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sentence_transformers import SentenceTransformer
from sklearn.decomposition import TruncatedSVD


# Step 1 - Extract embeddings
embedding_model = SentenceTransformer("all-MiniLM-L6-v2")
embeddings = embedding_model.encode(all_reviews)
```

```python
# Define potential cluster numbers
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]

# Initialize a list to store silhouette scores
silhouette_avg = []

# Iterate over different cluster numbers
for num_clusters in range_n_clusters:
    # Initialize KMeans model
    kmeans = KMeans(n_clusters=num_clusters)
    kmeans.fit(embeddings)

    # Get cluster labels
    cluster_labels = kmeans.labels_

    # Calculate silhouette score
    silhouette_avg.append(silhouette_score(embeddings, cluster_labels))

# Find the optimal number of clusters (highest silhouette score)
optimal_clusters = range_n_clusters[silhouette_avg.index(max(silhouette_avg))]
print(f"Optimal number of clusters: {optimal_clusters}")

# Plot silhouette scores
plt.plot(range_n_clusters, silhouette_avg, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Silhouette score')
plt.title('Silhouette analysis For Optimal k')
plt.show()
```

```
Optimal number of clusters: 2
```



As shown in this example, tthe most optimal number of clusters is 2.

5.3 Optimized model

This code re does the Bert topic modelling using 2 clusters which was deemed to be the most optimal

[56]:
```
from sentence_transformers import SentenceTransformer
from umap import UMAP
from sklearn.decomposition import TruncatedSVD, PCA
from sklearn.cluster import KMeans
from hdbscan import HDBSCAN
from sklearn.feature_extraction.text import CountVectorizer
from bertopic.vectorizers import ClassTfidfTransformer
from bertopic import BERTopic
from sklearn.cluster import DBSCAN

# Step 1 - Extract embeddings
embedding_model = SentenceTransformer("all-MiniLM-L6-v2")
```

```
# Step 2 - Reduce dimensionality
if RedDiMethod == 'SVD':
   RD_model = TruncatedSVD(n_components=20)
elif RedDiMethod == 'PCA':
   RD_model = PCA(n_components='mle')
else:
   RD_model = UMAP(n_neighbors=15, n_components=5, min_dist=0.0,␣
 ↪metric='cosine')


# Step 3 - Cluster reduced embeddings
Cluster_model = KMeans(n_clusters= 2, random_state=42)

# Step 4 - Tokenize topics
vectorizer_model = CountVectorizer(stop_words="english")

# Step 5 - Create topic representation
ctfidf_model = ClassTfidfTransformer()

# All steps together
topic_model = BERTopic(
  embedding_model=embedding_model,    # Step 1 - Extract embeddings
  umap_model=RD_model,                # Step 2 - Reduce dimensionality
  hdbscan_model=Cluster_model,        # Step 3 - Cluster reduced embeddings
  vectorizer_model=vectorizer_model,  # Step 4 - Tokenize topics
  ctfidf_model=ctfidf_model,          # Step 5 - Extract topic words
  calculate_probabilities=True
)

topics, probs = topic_model.fit_transform(all_reviews)
```

5.4 4C - Evaulation

The following blocks display different evaluation methods to analyse the performance if the topic model with 2 clusters

[57]:
```
freq = topic_model.get_topic_info(); freq.head(n_clusters)
```

[57]:
```
   Topic  Count                       Name  \
0      0     25   0_ring_love_great_bought
1      1     25  1_wearing_great_gift_item

                                      Representation  \
0  [ring, love, great, bought, gift, diamond, wea…
1  [wearing, great, gift, item, beautiful, lookin…

                                 Representative_Docs
```

```
0   [ring beautiful ring first shipment ring scrat…
1   [one beautiful rosary seen smoothness color be…
```

[58]:
```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

word_cloud_images = []

for cluster in range(2):
    words = topic_model.get_topic(cluster)
    words_list = [word[0] for word in words]

    wordcloud = WordCloud(background_color="white", max_words=5000,␣
 ↪contour_width=3, contour_color='steelblue')
    wordcloud.generate(' '.join(words_list))

    # Append the word cloud image to the list
    word_cloud_images.append(wordcloud.to_image())

# Display the word cloud images side by side
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

for i, image in enumerate(word_cloud_images):
    axes[i].imshow(image)
    axes[i].set_title(f'Cluster {i + 1} - Word Cloud')
    axes[i].axis('off')

plt.tight_layout()
plt.show()
```

[59]:
```
topic_model.get_topic(0)  # Select the most frequent topic
```

[59]:
```
[('ring', 0.31881683263936944),
 ('love', 0.12210406376395636),
 ('great', 0.08373956345985002),
```

```
('bought', 0.06730216602887347),
('gift', 0.054554811578092166),
('diamond', 0.053834016378258236),
('wear', 0.051778263722971195),
('look', 0.04535643590681279),
('like', 0.04535643590681279),
('beautiful', 0.04218311261495149)]
```

The following two visualisations do not show when converted to pdf format

[60]:
```
topic_model.visualize_hierarchy(top_n_topics=n_clusters)
```

[61]:
```
topic_model.visualize_barchart(top_n_topics=n_clusters)
```

5.5 4D & 4E - Interactive UI

Below combines the LSI model with the Bert topic modelling. The user can type in a query and the number of retrieved documents they would like to recieve. The algorithm then finds the optimal number of clusters from that data and then clusters into topics. Note the data is entering Bert topic uses the cleaned data but is converted back to the raw data for printing purposes. Graphs that show statistics about the number of sentences per cluster and the top keywords per cluster are also presented to the user in this code

[62]:
```
import ipywidgets as widgets
from IPython.display import clear_output
from sentence_transformers import SentenceTransformer
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

allw = widgets.Output()

reviews = []

def retrieve_top_docs(_):

    clear_output(wait=True)
    display(allw)
    query = query_field.value
    n = n_doc_field.value

    all_reviews = []

    # Check if query or n is empty
    if not query or not n:
```

```
        print("Please enter both a query and a value for the number of␣
↪documents.")
        return
    try:
        n = int(n)
        if n <= 0:
            print("Please enter a valid positive value for the number of␣
↪documents.")
            return
    except ValueError:
        print("Please enter a valid numeric value for the number of documents.")
        return


    # mapping from id to vectorized document
    id_to_vectorized = dict(zip(ids, data_vectorized))

    # Find similarities for each query
    processed_query = process_query(query, vocab, best_norm)

    transformed_query = trunc_SVD_model.transform(processed_query)
    similarities = cosine_similarity(data_vectorized, transformed_query)
    indexes = np.argsort(similarities.flat)[-n:]

    # Retrieve the top n vectorized documents
    top_n_vectorized = data_vectorized[indexes]

    # Map the vectorized documents back to their corresponding IDs
    top_n_ids = [id for id, vectorized in id_to_vectorized.items() if␣
↪vectorized in top_n_vectorized]

    for id in top_n_ids:
        raw_review = id_to_raw_data.get(id, None)
        cleaned_review = id_to_clean_data.get(id, None)

        all_reviews.append(cleaned_review)

    reviews.append(all_reviews)
    apply_clustering(all_reviews)

    return

def embed_model(reviews):

    embedding_model = SentenceTransformer("all-MiniLM-L6-v2")
    embeddings = embedding_model.encode(reviews)
```

```
    return embeddings


def optimize_clustering(reviews):

    print("Finding optimal number of clusters...")
    embeddings = embed_model(reviews)
    # Define the valid range for the number of clusters
    n_samples = len(reviews)
    valid_range_n_clusters = list(range(2, min(n_samples, 8) + 1))

    # Initialize a list to store silhouette scores
    silhouette_avg = []

    # Iterate over different cluster numbers in the valid range
    for num_clusters in valid_range_n_clusters:
        # Initialize KMeans model
        kmeans = KMeans(n_clusters=num_clusters)
        kmeans.fit(embeddings)

        # Get cluster labels
        cluster_labels = kmeans.labels_

        # Calculate silhouette score
        silhouette_avg.append(silhouette_score(embeddings, cluster_labels))

    # Find the optimal number of clusters (highest silhouette score)
    optimal_clusters = valid_range_n_clusters[silhouette_avg.
 ↪index(max(silhouette_avg))]

    return optimal_clusters

def apply_clustering(reviews):


    optimal_clusters = optimize_clustering(reviews)
    print("Number of clusters", optimal_clusters)

    # Step 1 - Extract embeddings
    embedding_model = SentenceTransformer("all-MiniLM-L6-v2")

    # Step 2 - Reduce dimensionality
    if RedDiMethod == 'SVD':
        RD_model = TruncatedSVD(n_components=50)
    elif RedDiMethod == 'PCA':
        RD_model = PCA(n_components='mle')
    else:
```

```
        RD_model = UMAP(n_neighbors=15, n_components=5, min_dist=0.0,␣
↪metric='cosine')

    # Step 3 - Cluster reduced embeddings
    Cluster_model = KMeans(n_clusters=optimal_clusters, random_state=42)

    # Step 4 - Tokenize topics
    vectorizer_model = CountVectorizer(stop_words="english")

    # Step 5 - Create topic representation
    ctfidf_model = ClassTfidfTransformer()

    # All steps together
    topic_model = BERTopic(
        embedding_model=embedding_model,  # Step 1 - Extract embeddings
        umap_model=RD_model,  # Step 2 - Reduce dimensionality
        hdbscan_model=Cluster_model,  # Step 3 - Cluster reduced embeddings
        vectorizer_model=vectorizer_model,  # Step 4 - Tokenize topics
        ctfidf_model=ctfidf_model,  # Step 5 - Extract topic words
        calculate_probabilities=True
    )

    topics, probs = topic_model.fit_transform(reviews)

    topic_info = topic_model.get_topic_info()

    # Get the document IDs for each cluster
    cluster_documents = {cluster_id: [] for cluster_id in␣
↪range(optimal_clusters)}

    for doc_id, cluster_id in enumerate(topics):
        cluster_documents[cluster_id].append(doc_id)

    number_of_sentences_per_cluster = []


    # Print the sentences for each document in each cluster
    for cluster_id, documents in cluster_documents.items():

        top_words = topic_model.get_topic(cluster_id)
        top_10_words = top_words[:10]
        words = [word for word, _ in top_10_words]
        scores =[score for _, score in top_10_words]
        print(f"\nCluster {cluster_id + 1}")

        s = 1
```

```
        for doc_id in documents:
            # convert from clean data to raw for printing
            rev = reviews[doc_id ]
            raw_data = clean_data_to_raw.get(rev, None)
            print(s, raw_data)
            s += 1

        print("-"* 200)
        num_of_sent = (s - 1)
        number_of_sentences_per_cluster.append(num_of_sent)

        plt.figure(figsize=(8, 4))
        plt.bar(words, scores, color='red')
        plt.xlabel('Words')
        plt.ylabel('Score')
        plt.title('Number of words in cluster {}'.format(cluster_id + 1))



    # Plotting the graph
    plt.figure(figsize=(8, 4))
    plt.bar(topic_info['Topic'] + 1, number_of_sentences_per_cluster)
    plt.xlabel('Cluster')
    plt.ylabel('Number of Sentences')
    plt.title('Number of Sentences per Cluster')
    plt.xticks(range(1, optimal_clusters + 1))
    plt.show()

    return
```

[63]:
```
# Search button
search_button = widgets.Button(description="Search")
search_button.on_click(retrieve_top_docs)

# query field
query_field = widgets.Text(description="Enter Query:")
n_doc_field = widgets.Text(description="Enter the number of related documents␣
 ↪you wish to retrieve:")

#H layout
fields = widgets.HBox([query_field, n_doc_field])

output = widgets.Output()

allw = widgets.VBox([fields, search_button, output])
```

[64]:
```
display(allw)
```

```
VBox(children=(HBox(children=(Text(value='This is really great',␣
 ↪description='Enter Query:'), Text(value='50',…
```

```
Finding optimal number of clusters…
Number of clusters 3

Cluster 1
1 Ring is way too small and looks like a toy when putting it on. I would not
recommend if you want a nice 1/2 carat ring.
2 its what i wanted :) but its not my favorite piercing of mine but i have to
wear the bioplast cuz i break out with certain metals
3 I got this ring for my birthday and I love it, I cannot imagine a woman not
adoring this ring.
4 This ring is awesome! I wear the ring on my index finger and it fits great!
Comfortable ring.
5 I bought this ring as a Valentine's gift but couldn't give it since the hands
holding the heart symbol would reverse whenever the ring spins (there are 4 of
them on this ring, alternating the facing direction). It's a cute ring so I just
kept it for myself.
6 my husband loves it only thing is you cant have this ring resized due to the
way the ring is made
7 They definitely help lessen your appitite, however my ears were sore after
wearing for about 3 hours and the next few days I tried to wear them off and on
and to increase the wearing time. If you have a good pain tolerance you may not
notice any discomfort, as for me my ears lobes were swollen and I had to stop
wearing them for 4 days.
8 i got 3 belly rings 3 tongue rings 3 lip rings and a 16g cbr. all the stuff i
got was cool, i love the pink peace sign tongue ring :) the only complaint is
that i got 2 yellow lip rings and im not much for yellow, overall very happy
9 my only wish on this ring is- I wish the cut potrion went all the way around
the ring. Other than that a great very comfortable ring.
10 Bought this ring as a thumb ring everyone notices it and comments about how
pretty and cool it looks then when I spin it they want to know where I got it I
would recommend this ring to any one also very comfy.
11 I bought this for my 17 year old daughter who had seen a small heart shaped
ring at the Coach store and liked the style. This one was much cheaper and ended
up being so much prettier! It's really an eye catching ring, doesn't look cheap
and sparkles beautifully. I am actually ordering her a second ring since she
will be leaving for college next month and is worried about losing it. I think
she'll like knowing there is a spare ring at home just in case and for the price
it's not a big deal. She wears this ring on her right ring finger instead of her
high school ring because she loves it that much. I would suggest this ring for
any fan of hearts and a little bit of bling!
12 This ring is exactly what I wanted, I actually bought another and it wasn't
quite as solidly made as this ring is. Thanks for the great ring!
13 Great ring! I tried a different one first but the individual rings were so
```

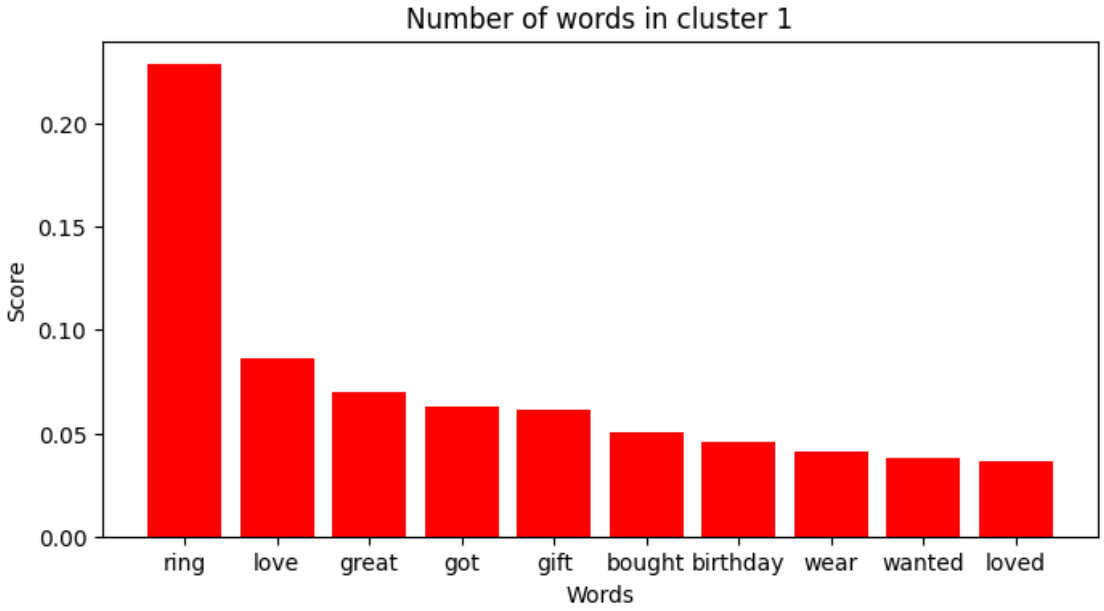
thin that they snapped within a few weeks. The rings on this one are thicker, making it a bolder and more durable ring. Of course it costs a bit more but you can't go wrong paying for better quality!
14 Recv'd my ring in a timely manner it looks very antique would recommend this ring to any garnet lover!
15 My second engagement Ring got it on my birthday i love this nice and shinny lol must buy ring for anyone
16 I got the ring as a promise ring for my girlfriend for Christmas and she loved it. Definitely a great value.
17 my wife loves the ring, it was a great gift. extremelly cheap and high quality.
18 This was a birthday gift for my 16 YO niece. She loves the ring and was very happy to have received it.
19 I love my birthstone and I wanted a piece of jewelry that symbolized the simple purity of the Blue Topaz. This ring did that for me. As a gift to myself for my birthday this year, it was definitely a great gift and a welcomed addition to my collection.
20 i got this ring as a gift from my boyfriend and i love it. the only thing is that if the rings are not position correctly it pinches the skin.
21 This ring has such a good sparkle and it looks like a ring that should cost 10x the amount. Makes a great gift for someone on a budget. My girlfriend loves it.
22 I bought this ring for my husband and he loved it. I received it when they said I would and it is a great ring
23 I love this ring fits just right and I showed my daughter the ring and she loved it as well. Great for everyday wear and the price was great..
24 I bought this as a gift for a friends birthday and she loved it. It's a beautifull ring.
25 I've owned several Claddagh rings over the years and lost my last one. I received this one as a gift recently and I truly love it. It's sturdy and the design is crisp. I can easily recommend this one as a lovely ring.
26 I have always wanted a claddaugh ring.This price was great.I love it
27 The diamond had a crack in one Garnet and another one had a large chip.
28 I really liked these earrings. However, i agree with one of the earlier reviews that I thought they would have been a little bigger.

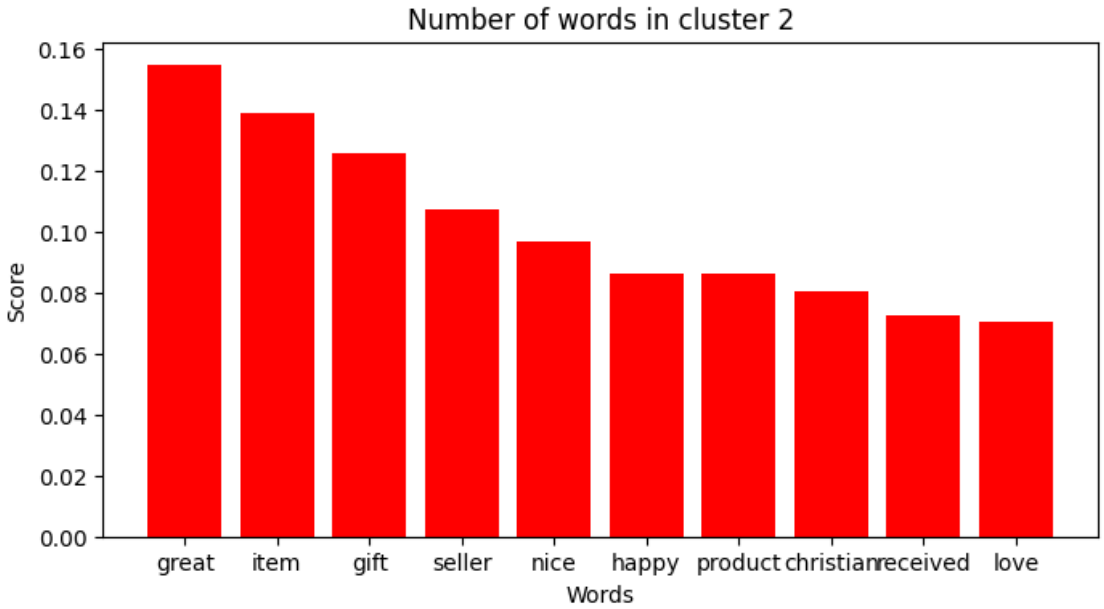
Cluster 2
1 I was very impressed with the quality and would not hesitate to purchase other items from the Seller. Their service was also exceptional.
2 Item was great quality and came promptly. I'm very happy with it and recommend it unreservedly.
3 A great gift to your loved one and an ever better seller. The seller deals with you in the most professional way and the security measures are superb.
4 this product made for a great gift and great memorize for my love and me. It

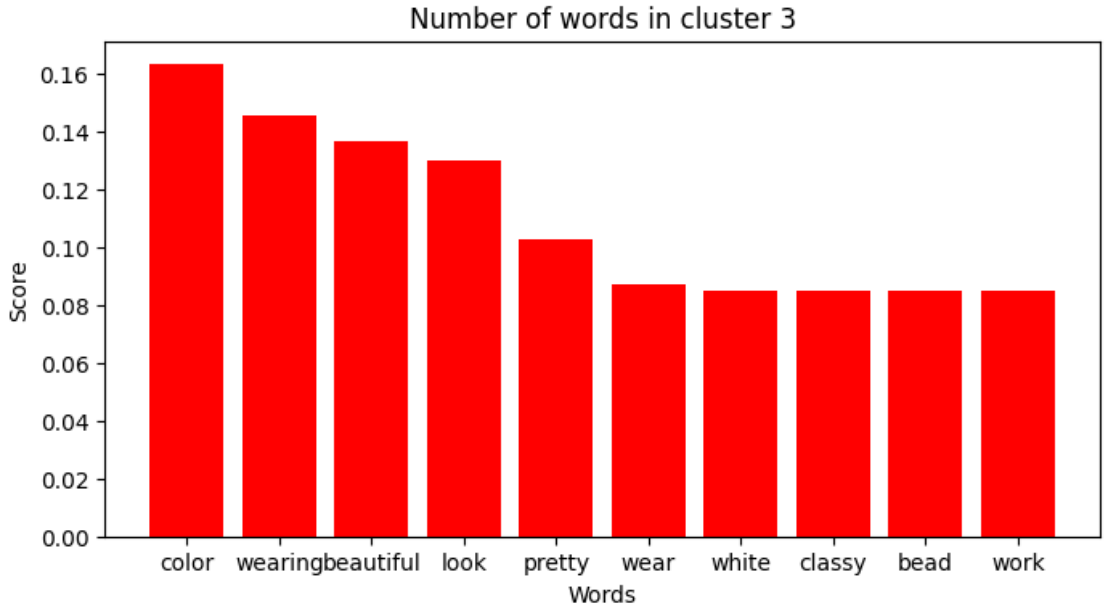
something we will always have. a helping gift from the heart that always shows you care.
5 My neice loves her birth stone so I got it for her for a Christmas Gift.I also love it also. great
6 I always love Willow Tree. they make great gifts for great people in your life. I have quite a collection, and I hope to continue to build it
7 My mother loved this and was a great birthday gift. These look even better in person and go great with anything.
8 I am happy with the product, I received it as advertised and in a timely manner; seller/Amazon kept me updated about shipment/delivery status. Would recommend item and seller
9 I am happy to say that though I did not receive my order the first time I ordered this cross, the company and Amazon did stand behind their product and refund the money spent on the item. Since they did, I re-ordered the item and it arrived in a timely manner. The cross is really nice. Thank you for the great customer service!
10 My item came quickly and in plenty of time for Christmas. They were a huge hit with the person who received them
11 I bought this to wear with my alice in wonderland costume for halloween, but I've been wearing it ever since I received it and have actually gotten a lot of compliments on it :)
12 The message is very positive and it looks very pretty. I bought it for my aunt as a present and the color is very nice.
13 I just got these yesterday as a Christmas gift- so far they look just like the picture and seem very nice.

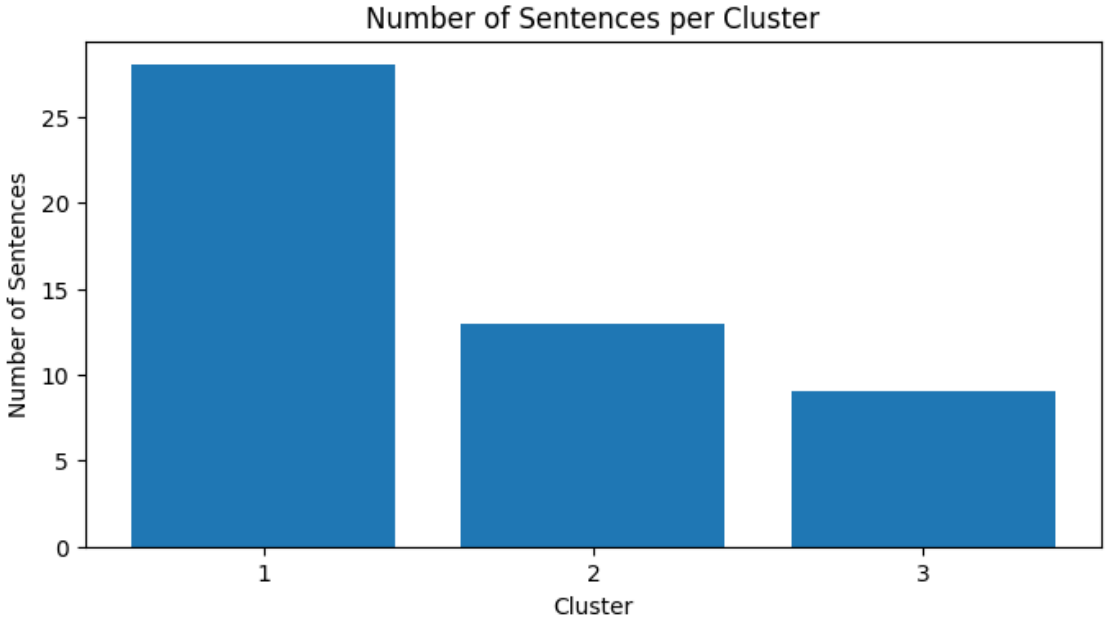
Cluster 3
1 The quality and look were not what I had anticipated. Very flimsy.I would not recommend this item
2 The days I do not wear the blue one I wear this one. I really enjoy wearing something Celtic and pretty.
3 I wanted a classy piece to wear on my right hand for work when I'm wearing Gold. I found that I will end up wearing this outside of work. Very classy looking
4 ery suitable for wearing for fashionable occasions. very dressy
5 great way to support the local pro sports team without wearing an oversized jersey or a hat to mess up the hair
6 The diamond looks pretty big. For the price, it shines brilliantly. The color doesn't look very white though. But you don't expect K color to be very white. Overall, I think it's pretty. and I am very happy with it.
7 These look quite like their photograph. They are very colorful and you know they are turtles. I've seen them elsewhere for quite a high price and these are beautiful.
8 You will not find a better deal for the money anywhere. This ring is so beautiful. The craftsmanship is meticulous and extensive. Also, the mussel

shells are pinker in person than they look online. They are deep in color in various shades of iridescent mauve.
9 The smaller carnelian beads have a clear beautiful orange color. It would be great if the elastic was the same color of the beads. Very beautiful.









6 Task 5 (Text Summarization)

This code below gets the reviews for each cluster created in Section 2 (Optimized model), converts back to raw text and combines the sentences ready for summarisation. Make sure to run the

'Optimized Model' code in Task 4 before running this cell

[81]:
```
all_sentences = []

# hard code optimal clusters for testing
optimal_clusters = 2

# Initialize a dictionary to store document IDs for each cluster
cluster_documents = {cluster_id: [] for cluster_id in range(optimal_clusters)}

# Populate the cluster_documents dictionary with document IDs
for doc_id, cluster_id in enumerate(topics):
    cluster_documents[cluster_id].append(doc_id)

# Print the sentences for each document in each cluster
for cluster_id, documents in cluster_documents.items():

    sentences_in_cluster = []
    for doc_id in documents:
        # Adjust document ID to match one-indexing
        doc_id += 1

        clean = all_reviews[doc_id - 1]  # Adjust index for zero-indexing
        sentence = clean_data_to_raw.get(clean, None)
        sentences_in_cluster.append(sentence)

    print(len((sentences_in_cluster)))
    cleaned = ' '.join(sentences_in_cluster)
    all_sentences.append(cleaned)

#print to view clustered reviews
#all_sentences
```

```
25
25
```

6.1 5A & 5B - Abstraction 1

[67]:
```
from transformers import BartTokenizer, PegasusTokenizer
from transformers import BartForConditionalGeneration,␣
 ↪PegasusForConditionalGeneration

IS_CNNDM = True # whether to use CNNDM dataset (BART-base) or XSum dataset␣
 ↪(PEGASUS-base)
LOWER = False

# Load our model checkpoints
if IS_CNNDM:
```

```
    brio_model = BartForConditionalGeneration.from_pretrained('Yale-LILY/
 ↪brio-cnndm-uncased')
    brio_tokenizer = BartTokenizer.from_pretrained('Yale-LILY/
 ↪brio-cnndm-uncased')
else:
    brio_model = PegasusForConditionalGeneration.from_pretrained('Yale-LILY/
 ↪brio-xsum-cased')
    brio_tokenizer = PegasusTokenizer.from_pretrained('Yale-LILY/
 ↪brio-xsum-cased')
```

```
config.json:   0%|          | 0.00/1.63k [00:00<?, ?B/s]
pytorch_model.bin:   0%|          | 0.00/1.63G [00:00<?, ?B/s]
tokenizer_config.json:   0%|          | 0.00/1.29k [00:00<?, ?B/s]
vocab.json:   0%|          | 0.00/899k [00:00<?, ?B/s]
merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]
special_tokens_map.json:   0%|          | 0.00/772 [00:00<?, ?B/s]
```

The following code use the pre-trained model 'Brio' to generate summaries for the 2 clusters. A title is also generated by using summary as the input and creating another summary from this.

[69]:
```
max_length = 100
title_length = 10

brio_summaries = []
for sentences in all_sentences:

    # Tokenize the text
    input_ids = brio_tokenizer.encode(sentences[:1024], return_tensors='pt')
    # Generate summary with the model
    summary_ids = brio_model.generate(input_ids, max_length=max_length)
    # Decode the summary
    summary = brio_tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    brio_summaries.append(summary)

    # Generate title from summary
    title_ids = brio_model.generate(summary_ids, max_length=title_length)
    title = brio_tokenizer.decode(title_ids[0], skip_special_tokens=True)

    print("Title: {}".format(title))
    print("Summary:")
    print(summary)

    print('-' * 100)
```

```
Title: first shipment of this ring had scratches
```

```
Summary:
first shipment of this ring had scratches on the diamond and was missing 2
diamonds on the shoulder of the ring. this ring is beyond gorgeous, I just love
it. I bought this ring as a Valentine's gift but couldn't give it to my husband
because it spins.
--------------------------------------------------------------------------------
--------------------
Title: spend your money on a higher
Summary:
spend your money on a higher quality item. the Holy Land Sand is present in a
compartment on the rosary. one of the jewels fell out of the wing and the metal
is already tarnishing. the bioplast helps lessen the appitite but your ears are
sore after wearing.
--------------------------------------------------------------------------------
--------------------
```

6.2 5A & 5B - Abstraction 2

[70]:
```
# install

%cd /content/
!git clone https://github.com/abertsch72/unlimiformer.git
!pip install -q -r unlimiformer/requirements.txt
!pip install -q faiss-cpu
%cd unlimiformer/src

from unlimiformer import Unlimiformer
from random_training_unlimiformer import RandomTrainingUnlimiformer
from usage import UnlimiformerArguments, training_addin

from transformers import BartForConditionalGeneration, AutoTokenizer
from datasets import load_dataset
import torch

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# example using govreport
modelname = "abertsch/unlimiformer-bart-govreport-alternating"

unlimiformer_tokenizer = AutoTokenizer.from_pretrained("facebook/bart-base")
unlimiformer_model = BartForConditionalGeneration.from_pretrained(modelname)

defaults = UnlimiformerArguments()
unlimiformer_kwargs = {
            'layer_begin': defaults.layer_begin,
            'layer_end': defaults.layer_end,
            'unlimiformer_head_num': defaults.unlimiformer_head_num,
```

```
            'exclude_attention': defaults.unlimiformer_exclude,
            'chunk_overlap': defaults.unlimiformer_chunk_overlap,
            'model_encoder_max_len': defaults.unlimiformer_chunk_size,
            'verbose': defaults.unlimiformer_verbose, 'tokenizer':␣
 ↪unlimiformer_tokenizer,
            'unlimiformer_training': defaults.unlimiformer_training,
            'use_datastore': defaults.use_datastore,
            'flat_index': defaults.flat_index,
            'test_datastore': defaults.test_datastore,
            'reconstruct_embeddings': defaults.reconstruct_embeddings,
            'gpu_datastore': defaults.gpu_datastore,
            'gpu_index': defaults.gpu_index
}

unlimiformer_model.to(device)

unlimiformer_model = Unlimiformer.convert_model(unlimiformer_model,␣
 ↪**unlimiformer_kwargs)
unlimiformer_model.eval()
unlimiformer_model.to(device)

clear_output()
```

[71]:
```
max_length = 100
title_length = 10

unilimiformer_summaries = []

for sentences in all_sentences:

  # Tokenize the text
  input_ids = unlimiformer_tokenizer.encode(sentences[:1024],␣
 ↪return_tensors='pt')
  # Generate summary with the model
  summary_ids = unlimiformer_model.generate(input_ids, max_length=max_length)
  # Decode the summary
  summary = unlimiformer_tokenizer.decode(summary_ids[0],␣
 ↪skip_special_tokens=True)
  unilimiformer_summaries.append(summary)

  # title
  title_ids = unlimiformer_tokenizer.encode(summary, return_tensors='pt')
  title_ids = unlimiformer_model.generate(input_ids, max_length=title_length)
  title = unlimiformer_tokenizer.decode(title_ids[0], skip_special_tokens=True)


  print("Title: {}".format(title ))
```

```
  print("Summmary:")
  print(summary)
  print('-' * 100)
```

```
INFO:Unlimiformer:Encoding 0 to 225 out of 225
INFO:Unlimiformer:Encoding 0 to 225 out of 225
INFO:Unlimiformer:Encoding 0 to 232 out of 232
```

```
Title: This ring is a beautiful ring but
Summmary:
This ring is a beautiful ring but the first shipment of this ring had scratches
on the diamond and was missing 2 diamonds on the shoulder of the ring. I
received this ring for a replacement of the same ring, the second ring was even
worse. Clearly visible white scratch right in the middle of a black diamond does
not look that great. I also bought this ring as a Valentine's gift but could not
give it since the hands holding the symbol would reverse whenever the ring
spins. It is
----------------------------------------------------------------------------------
--------------------
```

```
INFO:Unlimiformer:Encoding 0 to 232 out of 232
```

```
Title: Why GAO Did This Study

Summmary:
Why GAO Did This Study

This testimony discusses GAO's decision to purchase an piercing of mine which
was not mentioned in the description. The jewels fell out of the wing and one of
the jewels fell from the wing. GAO could see the quality work in this rosary.
There was only one thing buyers need to be aware of when buying this item that
wasn't mentioned in description, the Holy Land Sand is present in a compartment
on the rosaries. GA
----------------------------------------------------------------------------------
--------------------
```

6.3 5A & 5B - Extraction 1

[72]:
```
# install model
%cd /content/
!pip install -q bert-extractive-summarizer
from summarizer import Summarizer

bertsum_model = Summarizer()
```

```
/content
```

```
config.json:   0%|          | 0.00/571 [00:00<?, ?B/s]
```

```
model.safetensors:   0%|          | 0.00/1.34G [00:00<?, ?B/s]
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
```

[76]:
```python
# function to create title for extraction models using Brio

def generate_title(summary):

    title_length = 10
    # Generate title from summary
    title_ids = brio_model.generate(summary_ids, max_length=title_length)
    title = brio_tokenizer.decode(title_ids[0], skip_special_tokens=True)

    return title
```

For creating titles using an extracton method, this wasnt possible as you can only determine number of sentences you want rather than the length of the title, so when using an extraction method for this, the titles was way too long. The solution to this was by using an abstaction method as shown in the function above to create the titles

[77]:
```python
bertsum_summaries = []

for sentences in all_sentences:

  # create summary for each cluster
  summary = bertsum_model(sentences[:1024], num_sentences=10)
  bertsum_summaries.append(summary)

  # generate title for summary using brio
  title = generate_title(summary)

  print("Title: {}".format(title))
  print("Summary:")
  print(summary)
  print('-' * 100)
```

```
Title: GAO purchased a rosary
Summary:
This ring is a beautiful ring but the first shipment of this ring had scratches
on the diamond and was missing 2 diamonds on the shoulder of the ring. I
returned the ring for a replacement of the same ring, the second ring was even
worse. Clearly visible white scratch right in the middle of a black diamond does
not look that great. And this time 1 diamond missing on the shoulder of the
ring. I wanted to know if this ring is like 2 rings in one, because this ring is
beyond gorgeous, I just love it. I got this ring for my birthday and I love it,
I cannot imagine a woman not adoring this ring. I wear the ring on my index
finger and it fits great! I bought this ring as a Valentine's gift but couldn't
```

```
give it since the hands holding the heart symbol would reverse whenever the ring
spins (there are 4 of them on this ring, alternating the facing direction). It's
a cute ring so I just kept it for myself.
--------------------------------------------------------------------------------
--------------------
Title: GAO purchased a rosary
Summary:
I only had it a week before one of the jewels fell out of the wing. Spend your
money on a higher quality item. its what i wanted :) but its not my favorite
piercing of mine but i have to wear the bioplast cuz i break out with certain
metals They definitely help lessen your appitite, however my ears were sore
after wearing for about 3 hours and the next few days I tried to wear them off
and on and to increase the wearing time. If you have a good pain tolerance you
may not notice any discomfort, as for me my ears lobes were swollen and I had to
stop wearing them for 4 days. I could see the quality work in this rosary. I was
pleased when I opened the container and I could immediately smell olive wood.
There was only one thing buyers need to be aware of when buying this item that
wasn't mentioned in the description. The Holy Land Sand is present in a
compartment on the rosary. In my opinion, this adds to the quality of the rosary
so I'm not sure why the seller didn't describe
--------------------------------------------------------------------------------
--------------------
```

6.4 5A & 5B - Extraction 2

[78]:
```
# install model
%cd /content/
!pip install -U -q sentence-transformers
from summarizer.sbert import SBertSummarizer

sbert_model = SBertSummarizer('paraphrase-MiniLM-L6-v2')
```

```
/content
modules.json:   0%|          | 0.00/229 [00:00<?, ?B/s]
config_sentence_transformers.json:   0%|          | 0.00/122 [00:00<?, ?B/s]
README.md:   0%|          | 0.00/3.73k [00:00<?, ?B/s]
sentence_bert_config.json:   0%|          | 0.00/53.0 [00:00<?, ?B/s]
config.json:   0%|          | 0.00/629 [00:00<?, ?B/s]
pytorch_model.bin:   0%|          | 0.00/90.9M [00:00<?, ?B/s]
tokenizer_config.json:   0%|          | 0.00/314 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
special_tokens_map.json:   0%|          | 0.00/112 [00:00<?, ?B/s]
```

```
1_Pooling/config.json:   0%|          | 0.00/190 [00:00<?, ?B/s]
```

[79]:
```
j = 1
max_length = 100


sbert_summaries = []

for sentences in all_sentences:

  # create a summary for each cluster
  summary = sbert_model(sentences[:1024], num_sentences=5)
  sbert_summaries.append(summary)

  #generate title using brio
  title = generate_title(summary)

  print("Title: {}".format(title))
  print("Summary:")
  print(summary)
  print('-' * 100)

  j+=1
```

```
Title: GAO purchased a rosary
Summary:
This ring is a beautiful ring but the first shipment of this ring had scratches
on the diamond and was missing 2 diamonds on the shoulder of the ring. I
returned the ring for a replacement of the same ring, the second ring was even
worse. Clearly visible white scratch right in the middle of a black diamond does
not look that great. I got this ring for my birthday and I love it, I cannot
imagine a woman not adoring this ring. I wear the ring on my index finger and it
fits great!
----------------------------------------------------------------------------------
--------------------
Title: GAO purchased a rosary
Summary:
I only had it a week before one of the jewels fell out of the wing. Spend your
money on a higher quality item. its what i wanted :) but its not my favorite
piercing of mine but i have to wear the bioplast cuz i break out with certain
metals They definitely help lessen your appitite, however my ears were sore
after wearing for about 3 hours and the next few days I tried to wear them off
and on and to increase the wearing time. I was pleased when I opened the
container and I could immediately smell olive wood. In my opinion, this adds to
the quality of the rosary so I'm not sure why the seller didn't describe
----------------------------------------------------------------------------------
--------------------
```

6.5 5C - Evaluation with gpt summary

As can be observed above, the appear summarys perform well and are meangingful however, the titles stuggle to summarise the whole summary. This could be due to the lack of input to the title generator, as the summary itself was used for this. For futuretesting, finding a pre-trained model may be more suited for this task. To evaluate the performance of the model, chatGPT was asked to summarise the same text for both clusters as a reference. Both the chatGPT summaries and the summaries created from the models were embedded using the Distilbert embedding function in section 3a to gather the semantic meaning of the texts. Cosine similarity is then used between the two in order to gather the similary between the texts

[82]:
```
#GPT summaries
summary1 = "The ring initially received had scratches and missing diamonds. A␣
 ↪replacement was also flawed, prompting a refund. Despite flaws, the ring is␣
 ↪adored for its beauty and comfort. Some users found issues with resizing and␣
 ↪soft metal. However, many praised its appearance, sparkle, and value, with␣
 ↪some considering it two rings in one. It's recommended for its durability,␣
 ↪affordability, and suitability as a gift. Users appreciated its design,␣
 ↪particularly for lovers of hearts and Claddagh rings. Overall, despite some␣
 ↪quality concerns, it's seen as a worthwhile purchase for its aesthetics and␣
 ↪value."
summary2 = "The item received mixed reviews, with complaints about jewel␣
 ↪quality and metal tarnishing, while others found it satisfactory. Some␣
 ↪experienced discomfort with earrings, while a rosary received praise for its␣
 ↪craftsmanship and inclusion of Holy Land sand. Buyers appreciated jewelry␣
 ↪quality compared to price, considering it a great gift. Birthstone jewelry␣
 ↪was well-received, with sentiments of sentimentality and appreciation.␣
 ↪Various items, including Celtic pieces and sports-themed jewelry, were␣
 ↪praised for their appearance and suitability for occasions. Overall, while␣
 ↪some had issues, many found the items beautiful and worth the purchase."


# clean and embed summary 1
clean_gpt_1 = clean_data(summary1)
cleaned_gpt_1 = ' '.join(clean_gpt_1)
gptdocs_1 = pd.Series(cleaned_gpt_1)
embedGPT_1 = embed_data(gptdocs_1)

# clean and embed summary 2
clean_gpt_2 = clean_data(summary2)
cleaned_gpt_2 = ' '.join(clean_gpt_2)
gptdocs_2 = pd.Series(cleaned_gpt_2)
embedGPT_2 = embed_data(gptdocs_2)
```

[83]:
```
#Brio clean and embed
clean_summary_1 = clean_data(brio_summaries[0])
cleaned_summary_1 = ' '.join(clean_summary_1)
clean_docs_1 = pd.Series(cleaned_summary_1)
```

```
embeded_data_1 = embed_data(clean_docs_1)


clean_summary_2 = clean_data(brio_summaries[1])
cleaned_summary_2 = ' '.join(clean_summary_1)
clean_docs_2 = pd.Series(cleaned_summary_2)
embeded_data_2 = embed_data(clean_docs_2)

#Unilimiformer clean and embed
clean_summary_3 = clean_data(unilimiformer_summaries[0])
cleaned_summary_3 = ' '.join(clean_summary_3)
clean_docs_3 = pd.Series(cleaned_summary_3)
embeded_data_3 = embed_data(clean_docs_3)

clean_summary_4 = clean_data(unilimiformer_summaries[1])
cleaned_summary_4 = ' '.join(clean_summary_4)
clean_docs_4 = pd.Series(cleaned_summary_4)
embeded_data_4 = embed_data(clean_docs_4)

#Bertsum clean and embed
clean_summary_5 = clean_data(bertsum_summaries[0])
cleaned_summary_5 = ' '.join(clean_summary_5)
clean_docs_5 = pd.Series(cleaned_summary_5)
embeded_data_5 = embed_data(clean_docs_5)


clean_summary_6 = clean_data(bertsum_summaries[1])
cleaned_summary_6 = ' '.join(clean_summary_6)
clean_docs_6 = pd.Series(cleaned_summary_6)
embeded_data_6 = embed_data(clean_docs_6)

#SBert clean and embed

clean_summary_7 = clean_data(sbert_summaries[0])
cleaned_summary_7 = ' '.join(clean_summary_7)
clean_docs_7 = pd.Series(cleaned_summary_7)
embeded_data_7 = embed_data(clean_docs_7)

clean_summary_8 = clean_data(sbert_summaries[1])
cleaned_summary_8 = ' '.join(clean_summary_8)
clean_docs_8 = pd.Series(cleaned_summary_8)
embeded_data_8 = embed_data(clean_docs_8)
```

[84]:
```
# calculate the cosine similarity's for the GPT summaries and generated␣
 ↪summaries

# Brio
```

```
similarities_1 = cosine_similarity(embeded_data_1, embedGPT_1)
similarities_2 = cosine_similarity(embeded_data_2, embedGPT_2)

#Uni
similarities_3 = cosine_similarity(embeded_data_3, embedGPT_1)
similarities_4 = cosine_similarity(embeded_data_4, embedGPT_2)

#BertSum

similarities_5 = cosine_similarity(embeded_data_5, embedGPT_1)
similarities_6 = cosine_similarity(embeded_data_6 , embedGPT_2)

#Sbert

similarities_7 = cosine_similarity(embeded_data_7, embedGPT_1)
similarities_8 = cosine_similarity(embeded_data_8 , embedGPT_2)
```

[85]:
```
# Get the data out of the list for better viewing purposes

#Brio

similarities_1 = similarities_1[0][0]
similarities_2 = similarities_2[0][0]

print(similarities_1)
print(similarities_2)

#Uni
similarities_3 = similarities_3[0][0]
similarities_4 = similarities_4[0][0]

print(similarities_3)
print(similarities_4)


#Bertsum

similarities_5 = similarities_5[0][0]
similarities_6 = similarities_6[0][0]

print(similarities_5)
print(similarities_6)


#SBert
```

```
similarities_7 = similarities_7[0][0]
similarities_8 = similarities_8[0][0]

print(similarities_7)
print(similarities_8 )
```

```
0.8654574
0.84637165
0.8390846
0.92341083
0.8758114
0.92741686
0.8898261
0.9093284
```

[86]:
```
# Create DataFrame and show the results
sim = {
    'Abstract - Brio, Cluster 1': [similarities_1],
    'Abstract - Brio, Cluster 2': [similarities_2],
    'Abstract - Unilimiformer, Cluster 1': [similarities_3],
    'Abstract - Unilimiformer, Cluster 2': [similarities_4],
    'Extract - Bertsum, Cluster 1': [similarities_5],
    'Extract - Bertsum, Cluster 2': [similarities_6],
    'Extract - SBert, Cluster 1': [similarities_7],
    'Extract - SBert, Cluster 2': [similarities_8]
}

df = pd.DataFrame(sim)

df
```

[86]:
```
   Abstract - Brio, Cluster 1  Abstract - Brio, Cluster 2  \
0                    0.865457                    0.846372

   Abstract - Unilimiformer, Cluster 1  Abstract - Unilimiformer, Cluster 2  \
0                             0.839085                             0.923411

   Extract - Bertsum, Cluster 1  Extract - Bertsum, Cluster 2  \
0                      0.875811                      0.927417

   Extract - SBert, Cluster 1  Extract - SBert, Cluster 2
0                    0.889826                    0.909328
```

[87]:
```
# Calculate average similarities for each group
BrioAverage = np.mean([similarities_1, similarities_2])
UnilimiformerAverage = np.mean([similarities_3, similarities_4])
BertSumAverage = np.mean([similarities_5, similarities_6])
```

```
SBertAverage = np.mean([similarities_7, similarities_8])

# Print average similarities for each group
print("Brio Average:", BrioAverage)
print("Unilimiformer Average:", UnilimiformerAverage)
print("BertSum Average:", BertSumAverage)
print("SBert Average:", SBertAverage)
print("_"* 100)

# Calculate standard deviation for each group
BrioStd = np.std([similarities_1, similarities_2])
UnilimiformerStd = np.std([similarities_3, similarities_4])
BertSumStd = np.std([similarities_5, similarities_6])
SBertStd = np.std([similarities_7, similarities_8])

# Print standard deviation for each group
print("Brio Standard Deviation:", BrioStd)
print("Unilimiformer Standard Deviation:", UnilimiformerStd)
print("BertSum Standard Deviation:", BertSumStd)
print("SBert Standard Deviation:", SBertStd)

# Calculate overall average similarities for abstract and extractive methods
AbstractAverage = np.mean([similarities_1, similarities_2, similarities_3,␣
 ↪similarities_4])
ExtractiveAverage = np.mean([similarities_5, similarities_6, similarities_7,␣
 ↪similarities_8])

# Print overall average similarities for abstract and extractive methods
print("-" * 100)
print("Abstractive method overall average:", AbstractAverage)
print("Extractive method overall average:", ExtractiveAverage)
```

```
Brio Average: 0.85591453
Unilimiformer Average: 0.88124776
BertSum Average: 0.9016141
SBert Average: 0.89957726
----------------------------------------------------------------------------------
--------------------
Brio Standard Deviation: 0.009542882
Unilimiformer Standard Deviation: 0.042163104
BertSum Standard Deviation: 0.025802732
SBert Standard Deviation: 0.009751141
----------------------------------------------------------------------------------
--------------------
Abstractive method overall average: 0.8685812
Extractive method overall average: 0.9005957
```

As can be observed above, the extractive methods work slightly better than the abstractive meth-

ods however, both methods perform well with high accuracy. From the two abstactive methods, Unilimiformer performs slightly better than Brio. For extractive, Bertsum performed better than SBert.

6.6 5D & 5E - Interactive UI

This interactive method combines the LSI, Bert topic modelling and summarisation. The user can type in a query, choose the number of documents to retrieve, cluster using either a set amount of clusters or allow the algorithm to optimize using the silouhette method from a dropdown box, and then the user has a choice of summarising with either an abstractive method (unilimiformer) or an extractive method (bertsum). The result shows the list of raw reviews and summary for each cluster. Note that the unilimiformer takes a little while to produce the summaries. For the additional feature as required for question 5d, a feature has been added to this section. This feature allows the user to group the reviews by their rating rather than using the clustering algorithm.The reviews are displayed in ascending order of rating and a summary of the whole group is displayed. Code was also added to format the printing so that there is a margin and the review/summary width doesnt go past a certain width on the screen.

IMPORTANT - In order to run this code, ensure all the previous code blocks for the models have ran including distilbert, bert topic, abstractive method 2 (unilimiformer), extraction model 1(bertsum) etc.

[88]:
```
import ipywidgets as widgets
from sklearn.metrics import silhouette_score
from IPython.display import clear_output
from sentence_transformers import SentenceTransformer
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
import textwrap
from bertopic.vectorizers import ClassTfidfTransformer

allw = widgets.Output()

reviews = []

# function to clean and embed query
def process_query(query, vocab, weightScheme):
    query = clean_data(query)
    line = ' '.join(query)
    vectorizer = CountVectorizer(vocabulary=vocab)
    transformer = TfidfTransformer(norm=weightScheme, sublinear_tf= True)
    encoded = transformer.fit_transform(vectorizer.fit_transform([line]))
    return encoded

# function to embed data
```

```
def vectorize_data(data, vocab, weightScheme):
    vectorizer = CountVectorizer(vocabulary=vocab)
    transformer = TfidfTransformer(norm= weightScheme, sublinear_tf=True)
    data_transformed = transformer.fit_transform(vectorizer.fit_transform(data))
    return data_transformed

def retrieve_top_docs(query, n):

    all_reviews = []

    # mapping from id to vectorized document
    id_to_vectorized = dict(zip(ids, data_vectorized))

    # Find similarities for each query
    processed_query = process_query(query, vocab, best_norm)

    transformed_query = trunc_SVD_model.transform(processed_query)
    similarities = cosine_similarity(data_vectorized, transformed_query)
    indexes = np.argsort(similarities.flat)[-n:]

    # Retrieve the top n vectorized documents
    top_n_vectorized = data_vectorized[indexes]

    # Map the vectorized documents back to their corresponding IDs
    top_n_ids = [id for id, vectorized in id_to_vectorized.items() if␣
 ↪vectorized in top_n_vectorized]

    for id in top_n_ids:
        review = id_to_raw_data.get(id, None)
        cleaned_review = id_to_clean_data.get(id, None)
        all_reviews.append(cleaned_review)

    reviews.append(all_reviews)

    return all_reviews

def embed_model(reviews):

    embedding_model = SentenceTransformer("all-MiniLM-L6-v2")
    embeddings = embedding_model.encode(reviews)

    return embeddings

def group_by_ratings(reviews, cluster_model):
    # dictionary to store reviews grouped by ratings
    reviews_by_ratings = {}
```

```
    for review in reviews:
        # First convert the clean data back to raw
        raw_data = clean_data_to_raw.get(review)

        # Retrieve the rating for the raw data
        rating = id_to_rating.get(raw_data, "None")

        # Add the review to the corresponding rating group
        if rating not in reviews_by_ratings:
            reviews_by_ratings[rating] = [raw_data]
        else:
            reviews_by_ratings[rating].append(raw_data)

    # Sort the ratings in ascending order
    sorted_ratings = sorted(reviews_by_ratings.keys())


    # Print reviews grouped by ratings in ascending order
    for rating in sorted_ratings:
        print("\nRating:", rating)
        all_reviews_in_rating_cluster = []
        s =1
        for review in reviews_by_ratings[rating]:
            all_reviews_in_rating_cluster.append(review)
            # set the width of the review
            wrapped_review_data = textwrap.fill(review, width=100,␣
↪subsequent_indent="    ")
            print(s, "|",wrapped_review_data)
            s+=1

        # join the reviews into one sentence
        all_reviews_joined = ' '.join(all_reviews_in_rating_cluster)
        summary = summarise_cluster(all_reviews_joined, cluster_model)

        print("\nSummary of rating group:")
        # set the width of summary
        wrapped_summary = textwrap.fill(summary, width=100)
        print(wrapped_summary)

        print("-" * 100)


    return reviews_by_ratings



def optimize_clustering(reviews):
```

```
    print("Finding optimal number of clusters.. ")
    embeddings = embed_model(reviews)
    # Define the valid range for the number of clusters
    n_samples = len(reviews)
    valid_range_n_clusters = list(range(2, min(n_samples, 8) + 1))

    # Initialize a list to store silhouette scores
    silhouette_avg = []

    # Iterate over different cluster numbers in the valid range
    for num_clusters in valid_range_n_clusters:
        # Initialize KMeans model
        kmeans = KMeans(n_clusters=num_clusters)
        kmeans.fit(embeddings)

        # Get cluster labels
        cluster_labels = kmeans.labels_

        # Calculate silhouette score
        silhouette_avg.append(silhouette_score(embeddings, cluster_labels))

    # Find the optimal number of clusters (highest silhouette score)
    optimal_clusters = valid_range_n_clusters[silhouette_avg.
↪index(max(silhouette_avg))]

    return optimal_clusters

def summarise_cluster(sentences, cluster_model):

  max_length = 100

  # user has choice of either extractive or abstractive
  if cluster_model == 'Extractive':


    summary = bertsum_model(sentences[:1024], num_sentences=10)

  if cluster_model == 'Abstractive':

    # Tokenize the text
    input_ids = unlimiformer_tokenizer.encode(sentences[:1024],␣
↪return_tensors='pt')
    # Generate summary with the model
    summary_ids = unlimiformer_model.generate(input_ids, max_length=max_length)
    # Decode the summary
```

```
    summary = unlimiformer_tokenizer.decode(summary_ids[0],␣
↪skip_special_tokens=True)

  return summary

def clustering(reviews, n_clusters):

    # Step 1 - Extract embeddings
    embedding_model = SentenceTransformer("all-MiniLM-L6-v2")

    # Step 2 - Reduce dimensionality
    if RedDiMethod == 'SVD':
        RD_model = TruncatedSVD(n_components=50)
    elif RedDiMethod == 'PCA':
        RD_model = PCA(n_components='mle')
    else:
        RD_model = UMAP(n_neighbors=15, n_components=5, min_dist=0.0,␣
↪metric='cosine')

    # Step 3 - Cluster reduced embeddings
    Cluster_model = KMeans(n_clusters=n_clusters, random_state=42)

    # Step 4 - Tokenize topics
    vectorizer_model = CountVectorizer(stop_words="english")

    # Step 5 - Create topic representation
    ctfidf_model = ClassTfidfTransformer()

    # All steps together
    topic_model = BERTopic(
        embedding_model=embedding_model,  # Step 1 - Extract embeddings
        umap_model=RD_model,  # Step 2 - Reduce dimensionality
        hdbscan_model=Cluster_model,  # Step 3 - Cluster reduced embeddings
        vectorizer_model=vectorizer_model,  # Step 4 - Tokenize topics
        ctfidf_model=ctfidf_model,  # Step 5 - Extract topic words
        calculate_probabilities=True
    )

    topics, probs = topic_model.fit_transform(reviews)

    return topics

def perform_processing(reviews, num_of_clusters, cluster_model):

    # if statements on number of clusters depending on user input
    if (num_of_clusters) == '2':
        n_clusters = 2
```

```
    elif (num_of_clusters) == '3':
        n_clusters = 3
    elif (num_of_clusters) == '4':
        n_clusters = 4
    elif (num_of_clusters) == '5':
        n_clusters = 5
    elif num_of_clusters == 'Optimize':
        n_clusters = optimize_clustering(reviews)

    print("Number of clusters", n_clusters)

    topics = clustering(reviews, n_clusters)

    topic_info = topic_model.get_topic_info()

    # Get the document IDs for each cluster
    cluster_documents = {cluster_id: [] for cluster_id in range(n_clusters)}

    for doc_id, cluster_id in enumerate(topics):
        cluster_documents[cluster_id].append(doc_id)

    number_of_sentences_per_cluster = []

    # Print the sentences for each document in each cluster
    for cluster_id, documents in cluster_documents.items():

        print(f"\nCluster {cluster_id + 1}")
        s = 1

        all_sentences_per_cluster = []


        for doc_id in documents:
            # Convert from clean data to raw for printing and summarization
            rev = reviews[doc_id]
            raw_data = clean_data_to_raw.get(rev, None)

            # Set width of review
            wrapped_raw_data = textwrap.fill(raw_data, width=100,␣
↪subsequent_indent="    ")

            print(s, "|", wrapped_raw_data)

            all_sentences_per_cluster.append(raw_data)
            s += 1
```

```
        # apply summarisation
        sentences = ' '.join(all_sentences_per_cluster)
        summary = summarise_cluster(sentences, cluster_model)
        print("\nSummary of cluster:")
        # set width of summary
        wrapped_summary = textwrap.fill(summary, width=100)
        print(wrapped_summary)

        print("-" * 100)

        num_of_sent = (s - 1)
        number_of_sentences_per_cluster.append(num_of_sent)

    return


def process(_):

  clear_output(wait=True)
  display(allw)
  query = query_field.value
  n = n_doc_field.value
  num_of_clusters = cluster_dropdown.value
  cluster_model = cluster_model_dropdown.value
  is_ratings = ratings_checkbox.value


  # Check if query or n is empty
  if not query or not n:
      print("Please enter both a query and a value for the number of documents.
 ↪")
      return
  try:
      n = int(n)
      if n <= 0:
          print("Please enter a valid positive value for the number of␣
 ↪documents.")
          return
  except ValueError:
      print("Please enter a valid numeric value for the number of documents.")
      return

  #get top docuemnts
  reviews = retrieve_top_docs(query, n)

  # group by ratings
  if is_ratings == True:
```

```
    group_by_ratings(reviews, cluster_model)

  # apply clustering
  else:
    perform_processing(reviews, num_of_clusters, cluster_model)

  return


# SVD on data
data_transformed = vectorize_data(data, vocab, best_norm)
trunc_SVD_model = TruncatedSVD(n_components=best_n_comp)
data_vectorized = trunc_SVD_model.fit_transform(data_transformed)
```

[89]:
```
# Search button
search_button = widgets.Button(description="Search")
search_button.on_click(process)

# number of clusters dropdown
cluster_options = ['2', '3', '4', '5', 'Optimize']
cluster_dropdown = widgets.Dropdown(options=cluster_options, description='Num␣
 ↪of Clusters:')

# cluster model

cluster_model_options = ['Abstractive', 'Extractive']
cluster_model_dropdown = widgets.Dropdown(options=cluster_model_options,␣
 ↪description='Cluster Model:')

# query field
query_field = widgets.Text(description="Enter Query:")
n_doc_field = widgets.Text(description="Enter the number of related documents␣
 ↪you wish to retrieve:")

# rating checkbox

# Create a checkbox widget
ratings_checkbox = widgets.Checkbox(
    value=False,
    description='Group by Ratings',
    disabled=False
)

#H layout
query_fields = widgets.HBox([query_field, n_doc_field])
cluster_fields = widgets.HBox([cluster_dropdown, cluster_model_dropdown])
```

```
output = widgets.Output()

allw = widgets.VBox([query_fields, cluster_fields,ratings_checkbox,␣
 ↪search_button, output])
```

[90]:
```
display(allw)
```

```
VBox(children=(HBox(children=(Text(value='This is really great',␣
 ↪description='Enter Query:'), Text(value='20',…
```

```
Finding optimal number of clusters..
Number of clusters 3

Cluster 1
1 | my only wish on this ring is- I wish the cut potrion went all the way around
the ring. Other than
    that a great very comfortable ring.
2 | This ring is exactly what I wanted, I actually bought another and it wasn't
quite as solidly made as
    this ring is. Thanks for the great ring!
3 | Great ring! I tried a different one first but the individual rings were so
thin that they snapped
    within a few weeks. The rings on this one are thicker, making it a bolder
and more durable ring.
    Of course it costs a bit more but you can't go wrong paying for better
quality!
4 | I got the ring as a promise ring for my girlfriend for Christmas and she
loved it. Definitely a
    great value.
5 | my wife loves the ring, it was a great gift. extremelly cheap and high
quality.
6 | This ring has such a good sparkle and it looks like a ring that should cost
10x the amount. Makes a
    great gift for someone on a budget. My girlfriend loves it.
7 | I bought this ring for my husband and he loved it. I received it when they
said I would and it is a
    great ring
8 | I love this ring fits just right and I showed my daughter the ring and she
loved it as well. Great
    for everyday wear and the price was great..
9 | I bought this as a gift for a friends birthday and she loved it. It's a
beautifull ring.
10 | I have always wanted a claddaugh ring.This price was great.I love it

Summary of cluster:
my only wish on this ring is- I wish the cut potrion went all the way around the
ring. Other than
```

that a great very comfortable ring. This ring is exactly what I wanted, I actually bought another
and it wasn't quite as solidly made as this ring is. I tried a different one first but the
individual rings were so thin that they snapped within a few weeks. The rings on this one are
thicker, making it a bolder and more durable ring. Of course it costs a bit more but you can't go
wrong paying for better quality! I got the ring as a promise ring for my girlfriend for Christmas
and she loved it. my wife loves the ring, it was a great gift. Makes a great gift for someone on a
budget. I received it when they said I would and it is a great ring I love this r
---

Cluster 2
1 | Item was great quality and came promptly. I'm very happy with it and recommend it unreservedly.
2 | I love my birthstone and I wanted a piece of jewelry that symbolized the simple purity of the Blue
    Topaz. This ring did that for me. As a gift to myself for my birthday this year, it was
    definitely a great gift and a welcomed addition to my collection.
3 | A great gift to your loved one and an ever better seller. The seller deals with you in the most
    professional way and the security measures are superb.
4 | this product made for a great gift and great memorize for my love and me. It something we will
    always have. a helping gift from the heart that always shows you care.
5 | I always love Willow Tree. they make great gifts for great people in your life. I have quite a
    collection, and I hope to continue to build it
6 | My mother loved this and was a great birthday gift. These look even better in person and go great
    with anything.

Summary of cluster:
Item was great quality and came promptly. I'm very happy with it and recommend it unreservedly. I
love my birthstone and I wanted a piece of jewelry that symbolized the simple purity of the Blue
Topaz. As a gift to myself for my birthday this year, it was definitely a great gift and a welcomed
addition to my collection. A great gift to your loved one and an ever better seller. The seller
deals with you in the most professional way and the security measures are

```
superb. this product made
for a great gift and great memorize for my love and me. a helping gift from the
heart that always
shows you care. they make great gifts for great people in your life. These look
even better in
person and go great with anything.
--------------------------------------------------------------------------------
--------------------

Cluster 3
1 | I wanted a classy piece to wear on my right hand for work when I'm wearing
Gold. I found that I will
    end up wearing this outside of work. Very classy looking
2 | ery suitable for wearing for fashionable occasions. very dressy
3 | I am looking forward to wearing them as they sparkle and catch every eye at
my son's wedding on June
    30
4 | great way to support the local pro sports team without wearing an oversized
jersey or a hat to mess
    up the hair

Summary of cluster:
I wanted a classy piece to wear on my right hand for work when I'm wearing Gold.
I found that I will
end up wearing this outside of work. Very classy looking ery suitable for
wearing for fashionable
occasions. very dressy I am looking forward to wearing them as they sparkle and
catch every eye at
my son's wedding on June 30 great way to support the local pro sports team
without wearing an
oversized jersey or a hat to mess up the hair
--------------------------------------------------------------------------------
--------------------
```

7 Code to convert to PDF

[ ]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc

clear_output()
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
pandoc is already the newest version (2.9.2.1-3ubuntu2).
pandoc set to manually installed.
The following additional packages will be installed:
  dvisvgm fonts-droid-fallback fonts-lato fonts-lmodern fonts-noto-mono fonts-
```

```
texgyre
  fonts-urw-base35 libapache-pom-java libcommons-logging-java libcommons-parent-
java
  libfontbox-java libfontenc1 libgs9 libgs9-common libidn12 libijs-0.35
libjbig2dec0 libkpathsea6
  libpdfbox-java libptexenc1 libruby3.0 libsynctex2 libteckit0 libtexlua53
libtexluajit2 libwoff1
  libzzip-0-13 lmodern poppler-data preview-latex-style rake ruby ruby-net-
telnet ruby-rubygems
  ruby-webrick ruby-xmlrpc ruby3.0 rubygems-integration t1utils teckit tex-
common tex-gyre
  texlive-base texlive-binaries texlive-fonts-recommended texlive-latex-base
  texlive-latex-recommended texlive-pictures texlive-plain-generic tipa xfonts-
encodings
  xfonts-utils
Suggested packages:
  fonts-noto fonts-freefont-otf | fonts-freefont-ttf libavalon-framework-java
  libcommons-logging-java-doc libexcalibur-logkit-java liblog4j1.2-java poppler-
utils ghostscript
  fonts-japanese-mincho | fonts-ipafont-mincho fonts-japanese-gothic | fonts-
ipafont-gothic
  fonts-arphic-ukai fonts-arphic-uming fonts-nanum ri ruby-dev bundler debhelper
gv
  | postscript-viewer perl-tk xpdf | pdf-viewer xzdec texlive-fonts-recommended-
doc
  texlive-latex-base-doc python3-pygments icc-profiles libfile-which-perl
  libspreadsheet-parseexcel-perl texlive-latex-extra-doc texlive-latex-
recommended-doc
  texlive-luatex texlive-pstricks dot2tex prerex texlive-pictures-doc vprerex
default-jre-headless
  tipa-doc
The following NEW packages will be installed:
  dvisvgm fonts-droid-fallback fonts-lato fonts-lmodern fonts-noto-mono fonts-
texgyre
  fonts-urw-base35 libapache-pom-java libcommons-logging-java libcommons-parent-
java
  libfontbox-java libfontenc1 libgs9 libgs9-common libidn12 libijs-0.35
libjbig2dec0 libkpathsea6
  libpdfbox-java libptexenc1 libruby3.0 libsynctex2 libteckit0 libtexlua53
libtexluajit2 libwoff1
  libzzip-0-13 lmodern poppler-data preview-latex-style rake ruby ruby-net-
telnet ruby-rubygems
  ruby-webrick ruby-xmlrpc ruby3.0 rubygems-integration t1utils teckit tex-
common tex-gyre texlive
  texlive-base texlive-binaries texlive-fonts-recommended texlive-latex-base
texlive-latex-extra
  texlive-latex-recommended texlive-pictures texlive-plain-generic texlive-xetex
tipa
```

```
  xfonts-encodings xfonts-utils
0 upgraded, 55 newly installed, 0 to remove and 38 not upgraded.
Need to get 182 MB of archives.
After this operation, 572 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy/main amd64 fonts-droid-fallback all
1:6.0.1r16-1.1build1 [1,805 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy/main amd64 fonts-lato all 2.0-2.1
[2,696 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy/main amd64 poppler-data all
0.4.11-1 [2,171 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tex-common all 6.17
[33.7 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy/main amd64 fonts-urw-base35 all
20200910-1 [6,367 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libgs9-common
all 9.55.0~dfsg1-0ubuntu5.6 [751 kB]
Get:7 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libidn12 amd64
1.38-4ubuntu1 [60.0 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy/main amd64 libijs-0.35 amd64
0.35-15build2 [16.5 kB]
Get:9 http://archive.ubuntu.com/ubuntu jammy/main amd64 libjbig2dec0 amd64
0.19-3build2 [64.7 kB]
Get:10 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libgs9 amd64
9.55.0~dfsg1-0ubuntu5.6 [5,031 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libkpathsea6
amd64 2021.20210626.59705-1ubuntu0.1 [60.3 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy/main amd64 libwoff1 amd64
1.0.2-1build4 [45.2 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy/universe amd64 dvisvgm amd64
2.13.1-1 [1,221 kB]
Get:14 http://archive.ubuntu.com/ubuntu jammy/universe amd64 fonts-lmodern all
2.004.5-6.1 [4,532 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy/main amd64 fonts-noto-mono all
20201225-1build1 [397 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy/universe amd64 fonts-texgyre all
20180621-3.1 [10.2 MB]
Get:17 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libapache-pom-java
all 18-1 [4,720 B]
Get:18 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libcommons-parent-
java all 43-1 [10.8 kB]
Get:19 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libcommons-logging-
java all 1.2-2 [60.3 kB]
Get:20 http://archive.ubuntu.com/ubuntu jammy/main amd64 libfontenc1 amd64
1:1.1.4-1build3 [14.7 kB]
Get:21 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libptexenc1
amd64 2021.20210626.59705-1ubuntu0.1 [39.1 kB]
Get:22 http://archive.ubuntu.com/ubuntu jammy/main amd64 rubygems-integration
all 1.18 [5,336 B]
```

```
Get:23 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 ruby3.0 amd64
3.0.2-7ubuntu2.4 [50.1 kB]
Get:24 http://archive.ubuntu.com/ubuntu jammy/main amd64 ruby-rubygems all
3.3.5-2 [228 kB]
Get:25 http://archive.ubuntu.com/ubuntu jammy/main amd64 ruby amd64 1:3.0~exp1
[5,100 B]
Get:26 http://archive.ubuntu.com/ubuntu jammy/main amd64 rake all 13.0.6-2 [61.7
kB]
Get:27 http://archive.ubuntu.com/ubuntu jammy/main amd64 ruby-net-telnet all
0.1.1-2 [12.6 kB]
Get:28 http://archive.ubuntu.com/ubuntu jammy/universe amd64 ruby-webrick all
1.7.0-3 [51.8 kB]
Get:29 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 ruby-xmlrpc all
0.3.2-1ubuntu0.1 [24.9 kB]
Get:30 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libruby3.0
amd64 3.0.2-7ubuntu2.4 [5,113 kB]
Get:31 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libsynctex2
amd64 2021.20210626.59705-1ubuntu0.1 [55.5 kB]
Get:32 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libteckit0 amd64
2.5.11+ds1-1 [421 kB]
Get:33 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libtexlua53
amd64 2021.20210626.59705-1ubuntu0.1 [120 kB]
Get:34 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libtexluajit2
amd64 2021.20210626.59705-1ubuntu0.1 [267 kB]
Get:35 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libzzip-0-13 amd64
0.13.72+dfsg.1-1.1 [27.0 kB]
Get:36 http://archive.ubuntu.com/ubuntu jammy/main amd64 xfonts-encodings all
1:1.0.5-0ubuntu2 [578 kB]
Get:37 http://archive.ubuntu.com/ubuntu jammy/main amd64 xfonts-utils amd64
1:7.7+6build2 [94.6 kB]
Get:38 http://archive.ubuntu.com/ubuntu jammy/universe amd64 lmodern all
2.004.5-6.1 [9,471 kB]
Get:39 http://archive.ubuntu.com/ubuntu jammy/universe amd64 preview-latex-style
all 12.2-1ubuntu1 [185 kB]
Get:40 http://archive.ubuntu.com/ubuntu jammy/main amd64 t1utils amd64
1.41-4build2 [61.3 kB]
Get:41 http://archive.ubuntu.com/ubuntu jammy/universe amd64 teckit amd64
2.5.11+ds1-1 [699 kB]
Get:42 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tex-gyre all
20180621-3.1 [6,209 kB]
Get:43 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 texlive-
binaries amd64 2021.20210626.59705-1ubuntu0.1 [9,848 kB]
Get:44 http://archive.ubuntu.com/ubuntu jammy/universe amd64 texlive-base all
2021.20220204-1 [21.0 MB]
Get:45 http://archive.ubuntu.com/ubuntu jammy/universe amd64 texlive-fonts-
recommended all 2021.20220204-1 [4,972 kB]
Get:46 http://archive.ubuntu.com/ubuntu jammy/universe amd64 texlive-latex-base
all 2021.20220204-1 [1,128 kB]
```

```
Get:47 http://archive.ubuntu.com/ubuntu jammy/universe amd64 texlive-latex-
recommended all 2021.20220204-1 [14.4 MB]
Get:48 http://archive.ubuntu.com/ubuntu jammy/universe amd64 texlive all
2021.20220204-1 [14.3 kB]
Get:49 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libfontbox-java all
1:1.8.16-2 [207 kB]
Get:50 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libpdfbox-java all
1:1.8.16-2 [5,199 kB]
Get:51 http://archive.ubuntu.com/ubuntu jammy/universe amd64 texlive-pictures
all 2021.20220204-1 [8,720 kB]
Get:52 http://archive.ubuntu.com/ubuntu jammy/universe amd64 texlive-latex-extra
all 2021.20220204-1 [13.9 MB]
Get:53 http://archive.ubuntu.com/ubuntu jammy/universe amd64 texlive-plain-
generic all 2021.20220204-1 [27.5 MB]
Get:54 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tipa all 2:1.3-21
[2,967 kB]
Get:55 http://archive.ubuntu.com/ubuntu jammy/universe amd64 texlive-xetex all
2021.20220204-1 [12.4 MB]
Fetched 182 MB in 14s (12.9 MB/s)
Extracting templates from packages: 100%
Preconfiguring packages …
Selecting previously unselected package fonts-droid-fallback.
(Reading database … 121752 files and directories currently installed.)
Preparing to unpack …/00-fonts-droid-fallback_1%3a6.0.1r16-1.1build1_all.deb
…
Unpacking fonts-droid-fallback (1:6.0.1r16-1.1build1) …
Selecting previously unselected package fonts-lato.
Preparing to unpack …/01-fonts-lato_2.0-2.1_all.deb …
Unpacking fonts-lato (2.0-2.1) …
Selecting previously unselected package poppler-data.
Preparing to unpack …/02-poppler-data_0.4.11-1_all.deb …
Unpacking poppler-data (0.4.11-1) …
Selecting previously unselected package tex-common.
Preparing to unpack …/03-tex-common_6.17_all.deb …
Unpacking tex-common (6.17) …
Selecting previously unselected package fonts-urw-base35.
Preparing to unpack …/04-fonts-urw-base35_20200910-1_all.deb …
Unpacking fonts-urw-base35 (20200910-1) …
Selecting previously unselected package libgs9-common.
Preparing to unpack …/05-libgs9-common_9.55.0~dfsg1-0ubuntu5.6_all.deb …
Unpacking libgs9-common (9.55.0~dfsg1-0ubuntu5.6) …
Selecting previously unselected package libidn12:amd64.
Preparing to unpack …/06-libidn12_1.38-4ubuntu1_amd64.deb …
Unpacking libidn12:amd64 (1.38-4ubuntu1) …
Selecting previously unselected package libijs-0.35:amd64.
Preparing to unpack …/07-libijs-0.35_0.35-15build2_amd64.deb …
Unpacking libijs-0.35:amd64 (0.35-15build2) …
Selecting previously unselected package libjbig2dec0:amd64.
```

```
Preparing to unpack …/08-libjbig2dec0_0.19-3build2_amd64.deb …
Unpacking libjbig2dec0:amd64 (0.19-3build2) …
Selecting previously unselected package libgs9:amd64.
Preparing to unpack …/09-libgs9_9.55.0~dfsg1-0ubuntu5.6_amd64.deb …
Unpacking libgs9:amd64 (9.55.0~dfsg1-0ubuntu5.6) …
Selecting previously unselected package libkpathsea6:amd64.
Preparing to unpack …/10-libkpathsea6_2021.20210626.59705-1ubuntu0.1_amd64.deb
…
Unpacking libkpathsea6:amd64 (2021.20210626.59705-1ubuntu0.1) …
Selecting previously unselected package libwoff1:amd64.
Preparing to unpack …/11-libwoff1_1.0.2-1build4_amd64.deb …
Unpacking libwoff1:amd64 (1.0.2-1build4) …
Selecting previously unselected package dvisvgm.
Preparing to unpack …/12-dvisvgm_2.13.1-1_amd64.deb …
Unpacking dvisvgm (2.13.1-1) …
Selecting previously unselected package fonts-lmodern.
Preparing to unpack …/13-fonts-lmodern_2.004.5-6.1_all.deb …
Unpacking fonts-lmodern (2.004.5-6.1) …
Selecting previously unselected package fonts-noto-mono.
Preparing to unpack …/14-fonts-noto-mono_20201225-1build1_all.deb …
Unpacking fonts-noto-mono (20201225-1build1) …
Selecting previously unselected package fonts-texgyre.
Preparing to unpack …/15-fonts-texgyre_20180621-3.1_all.deb …
Unpacking fonts-texgyre (20180621-3.1) …
Selecting previously unselected package libapache-pom-java.
Preparing to unpack …/16-libapache-pom-java_18-1_all.deb …
Unpacking libapache-pom-java (18-1) …
Selecting previously unselected package libcommons-parent-java.
Preparing to unpack …/17-libcommons-parent-java_43-1_all.deb …
Unpacking libcommons-parent-java (43-1) …
Selecting previously unselected package libcommons-logging-java.
Preparing to unpack …/18-libcommons-logging-java_1.2-2_all.deb …
Unpacking libcommons-logging-java (1.2-2) …
Selecting previously unselected package libfontenc1:amd64.
Preparing to unpack …/19-libfontenc1_1%3a1.1.4-1build3_amd64.deb …
Unpacking libfontenc1:amd64 (1:1.1.4-1build3) …
Selecting previously unselected package libptexenc1:amd64.
Preparing to unpack …/20-libptexenc1_2021.20210626.59705-1ubuntu0.1_amd64.deb
…
Unpacking libptexenc1:amd64 (2021.20210626.59705-1ubuntu0.1) …
Selecting previously unselected package rubygems-integration.
Preparing to unpack …/21-rubygems-integration_1.18_all.deb …
Unpacking rubygems-integration (1.18) …
Selecting previously unselected package ruby3.0.
Preparing to unpack …/22-ruby3.0_3.0.2-7ubuntu2.4_amd64.deb …
Unpacking ruby3.0 (3.0.2-7ubuntu2.4) …
Selecting previously unselected package ruby-rubygems.
Preparing to unpack …/23-ruby-rubygems_3.3.5-2_all.deb …
```

```
Unpacking ruby-rubygems (3.3.5-2) …
Selecting previously unselected package ruby.
Preparing to unpack …/24-ruby_1%3a3.0~exp1_amd64.deb …
Unpacking ruby (1:3.0~exp1) …
Selecting previously unselected package rake.
Preparing to unpack …/25-rake_13.0.6-2_all.deb …
Unpacking rake (13.0.6-2) …
Selecting previously unselected package ruby-net-telnet.
Preparing to unpack …/26-ruby-net-telnet_0.1.1-2_all.deb …
Unpacking ruby-net-telnet (0.1.1-2) …
Selecting previously unselected package ruby-webrick.
Preparing to unpack …/27-ruby-webrick_1.7.0-3_all.deb …
Unpacking ruby-webrick (1.7.0-3) …
Selecting previously unselected package ruby-xmlrpc.
Preparing to unpack …/28-ruby-xmlrpc_0.3.2-1ubuntu0.1_all.deb …
Unpacking ruby-xmlrpc (0.3.2-1ubuntu0.1) …
Selecting previously unselected package libruby3.0:amd64.
Preparing to unpack …/29-libruby3.0_3.0.2-7ubuntu2.4_amd64.deb …
Unpacking libruby3.0:amd64 (3.0.2-7ubuntu2.4) …
Selecting previously unselected package libsynctex2:amd64.
Preparing to unpack …/30-libsynctex2_2021.20210626.59705-1ubuntu0.1_amd64.deb
…
Unpacking libsynctex2:amd64 (2021.20210626.59705-1ubuntu0.1) …
Selecting previously unselected package libteckit0:amd64.
Preparing to unpack …/31-libteckit0_2.5.11+ds1-1_amd64.deb …
Unpacking libteckit0:amd64 (2.5.11+ds1-1) …
Selecting previously unselected package libtexlua53:amd64.
Preparing to unpack …/32-libtexlua53_2021.20210626.59705-1ubuntu0.1_amd64.deb
…
Unpacking libtexlua53:amd64 (2021.20210626.59705-1ubuntu0.1) …
Selecting previously unselected package libtexluajit2:amd64.
Preparing to unpack
…/33-libtexluajit2_2021.20210626.59705-1ubuntu0.1_amd64.deb …
Unpacking libtexluajit2:amd64 (2021.20210626.59705-1ubuntu0.1) …
Selecting previously unselected package libzzip-0-13:amd64.
Preparing to unpack …/34-libzzip-0-13_0.13.72+dfsg.1-1.1_amd64.deb …
Unpacking libzzip-0-13:amd64 (0.13.72+dfsg.1-1.1) …
Selecting previously unselected package xfonts-encodings.
Preparing to unpack …/35-xfonts-encodings_1%3a1.0.5-0ubuntu2_all.deb …
Unpacking xfonts-encodings (1:1.0.5-0ubuntu2) …
Selecting previously unselected package xfonts-utils.
Preparing to unpack …/36-xfonts-utils_1%3a7.7+6build2_amd64.deb …
Unpacking xfonts-utils (1:7.7+6build2) …
Selecting previously unselected package lmodern.
Preparing to unpack …/37-lmodern_2.004.5-6.1_all.deb …
Unpacking lmodern (2.004.5-6.1) …
Selecting previously unselected package preview-latex-style.
Preparing to unpack …/38-preview-latex-style_12.2-1ubuntu1_all.deb …
```

```
Unpacking preview-latex-style (12.2-1ubuntu1) …
Selecting previously unselected package t1utils.
Preparing to unpack …/39-t1utils_1.41-4build2_amd64.deb …
Unpacking t1utils (1.41-4build2) …
Selecting previously unselected package teckit.
Preparing to unpack …/40-teckit_2.5.11+ds1-1_amd64.deb …
Unpacking teckit (2.5.11+ds1-1) …
Selecting previously unselected package tex-gyre.
Preparing to unpack …/41-tex-gyre_20180621-3.1_all.deb …
Unpacking tex-gyre (20180621-3.1) …
Selecting previously unselected package texlive-binaries.
Preparing to unpack …/42-texlive-
binaries_2021.20210626.59705-1ubuntu0.1_amd64.deb …
Unpacking texlive-binaries (2021.20210626.59705-1ubuntu0.1) …
Selecting previously unselected package texlive-base.
Preparing to unpack …/43-texlive-base_2021.20220204-1_all.deb …
Unpacking texlive-base (2021.20220204-1) …
Selecting previously unselected package texlive-fonts-recommended.
Preparing to unpack …/44-texlive-fonts-recommended_2021.20220204-1_all.deb …
Unpacking texlive-fonts-recommended (2021.20220204-1) …
Selecting previously unselected package texlive-latex-base.
Preparing to unpack …/45-texlive-latex-base_2021.20220204-1_all.deb …
Unpacking texlive-latex-base (2021.20220204-1) …
Selecting previously unselected package texlive-latex-recommended.
Preparing to unpack …/46-texlive-latex-recommended_2021.20220204-1_all.deb …
Unpacking texlive-latex-recommended (2021.20220204-1) …
```

[18]:
```
!jupyter nbconvert --to pdf <'/content/drive/MyDrive/Colab Notebooks/
↪NLP_CW_F328685_George_Brown.ipynb.ipynb'>.ipynb
```

```
/bin/bash: line 1: /content/drive/MyDrive/Colab
Notebooks/George_Brown_F328685_NLP_CW.ipynb.ipynb: No such file or directory
```

[19]:
```
from google.colab import files
import os

# Define the notebook path
notebook_path = '/content/drive/MyDrive/Colab Notebooks/
↪NLP_CW_F328685_George_Brown.ipynb'

# Define the output directory
output_dir = '/content/'

# Convert the notebook to PDF
!jupyter nbconvert --to pdf "$notebook_path" --output-dir="$output_dir"

# Define the path to the PDF file
```

```python
pdf_file_path = '/content/NLP_CW_F328685_George_Brown.pdf'

# Check if the PDF file exists
if os.path.exists(pdf_file_path):
    # Download the PDF file
    files.download(pdf_file_path)
else:
    print("PDF file not found.")
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab
Notebooks/George_Brown_F328685_NLP_CW.ipynb to pdf
/usr/local/lib/python3.10/dist-packages/nbconvert/filters/datatypefilter.py:41:
UserWarning: Your element with mimetype(s) dict_keys(['text/html']) is not able
to be represented.
  warn(
/usr/local/lib/python3.10/dist-packages/nbconvert/filters/datatypefilter.py:41:
UserWarning: Your element with mimetype(s) dict_keys(['text/html']) is not able
to be represented.
  warn(
[NbConvertApp] Support files will be in George_Brown_F328685_NLP_CW_files/
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Making directory ./George_Brown_F328685_NLP_CW_files
[NbConvertApp] Writing 336870 bytes to notebook.tex
[NbConvertApp] Building PDF
Traceback (most recent call last):
  File "/usr/local/bin/jupyter-nbconvert", line 8, in <module>
    sys.exit(main())
  File "/usr/local/lib/python3.10/dist-packages/jupyter_core/application.py",
line 283, in launch_instance
    super().launch_instance(argv=argv, **kwargs)
  File "/usr/local/lib/python3.10/dist-
packages/traitlets/config/application.py", line 992, in launch_instance
    app.start()
```

```
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/nbconvertapp.py", line
423, in start
    self.convert_notebooks()
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/nbconvertapp.py", line
597, in convert_notebooks
    self.convert_single_notebook(notebook_filename)
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/nbconvertapp.py", line
560, in convert_single_notebook
    output, resources = self.export_single_notebook(
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/nbconvertapp.py", line
488, in export_single_notebook
    output, resources = self.exporter.from_filename(
  File "/usr/local/lib/python3.10/dist-
packages/nbconvert/exporters/exporter.py", line 189, in from_filename
    return self.from_file(f, resources=resources, **kw)
  File "/usr/local/lib/python3.10/dist-
packages/nbconvert/exporters/exporter.py", line 206, in from_file
    return self.from_notebook_node(
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/exporters/pdf.py",
line 194, in from_notebook_node
    self.run_latex(tex_file)
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/exporters/pdf.py",
line 164, in run_latex
    return self.run_command(
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/exporters/pdf.py",
line 111, in run_command
    raise OSError(
OSError: xelatex not found on PATH, if you have not installed xelatex you may
need to do so. Find further instructions at
https://nbconvert.readthedocs.io/en/latest/install.html#installing-tex.
PDF file not found.
```

[19]: