task1

August 12, 2024

1 Task1: Implementation of Tokenization, Stemming, Stop words removal

1.1 using user defined functions

[1]:
```
class week1:
    def tokenization(self, doc):
        tokens = []
        if '.' in doc:
            tokens.append('.')
            doc = doc.replace('.','')
        if ',' in doc:
            tokens.append(',')
            doc = doc.replace(',','')
        tokens.extend(doc.split())
        print("tokens: ", tokens)
        return tokens

    def stop_word(self,tokens):

        stop_words = [ "a", "an", "and", "are", "as", "at", "be", "but", "by",
                       "for", "if", "in", "into", "is", "it", "no", "not",␣
↪"of",
                       "on", "or", "such", "that", "the", "their", "then",␣
↪"there",
                       "these", "they", "this", "to", "was", "will", "with"]
        removed_stop_words = []
        for word in tokens:
            if word in stop_words:
                continue
            else:
                removed_stop_words.append(word)

        print("\ntokens after removing stop_words: ",removed_stop_words)
        return removed_stop_words

    def stemming(self,tokens):
```

```
        st = ['ed', 'ing', 'es']
        for i in range(len(tokens)):
            if tokens[i][-2:] == 'ed' or tokens[i][-2:] == 'es':
                tokens[i] = tokens[i][:-2]

            if tokens[i][-3:] == 'ing':
                tokens[i] = tokens[i][:-3]

            if tokens[i][-1:] == 's' and tokens[i][-2:] != 'is':
                tokens[i] = tokens[i][:-1]
        print("\nTokens after stemming: ", tokens)



obj = week1()
s = "This is IR lab. We are doing lab experiments"
tokens = obj.tokenization(s)
tokens = obj.stop_word(tokens)
obj.stemming(tokens)
```

```
tokens:  ['.', 'This', 'is', 'IR', 'lab', 'We', 'are', 'doing', 'lab',
'experiments']

tokens after removing stop_words:  ['.', 'This', 'IR', 'lab', 'We', 'doing',
'lab', 'experiments']

Tokens after stemming:  ['.', 'This', 'IR', 'lab', 'We', 'do', 'lab',
'experiment']
```

task2

August 12, 2024

1 Task1: Implementation of Tokenization, Stemming, Stop words removal

[1]:
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
```

1.1 Using NLTK

[2]:
```
class lab1:
    def tokenization(self, doc):
        tokens = word_tokenize(doc)
        return tokens

    def stop_word(self, tokens):
        stop_words = set(stopwords.words('english'))
        filtered_tokens = [word for word in tokens if word.lower() not in␣
 ↪stop_words]
        return filtered_tokens

    def stemming(self, filtered_tokens):
        stemmer = PorterStemmer()
        stemmed_tokens = [stemmer.stem(word) for word in filtered_tokens]
        return stemmed_tokens
```

[3]:
```
doc1 = open('IR.txt', 'r').read()
doc2 = open('IT.txt', 'r').read()
doc3 = open('jungle.txt', 'r').read()
documents = [doc1, doc2, doc3]
```

[4]:
```
processor = lab1()
i = 1
for doc in documents:
    tokens = processor.tokenization(doc)
    removed_stop_words = processor.stop_word(tokens)
    stemmed_tokens = processor.stemming(removed_stop_words)
```

```
    print(f"\n++++++++++ Document {i} ++++++++++++:")
    print("***tokens:*** \n", tokens)
    print("\n***tokens after removing stop_words: ***\n",removed_stop_words)
    print("\n***Tokens after stemming: ***\n", tokens)
 ␣
↪print("================================================================================")
    i+=1
```

```
++++++++++ Document 1 ++++++++++++:
***tokens:***
 ['Information', 'retrieval', 'is', 'the', 'process', 'of', 'obtaining',
'information', 'from', 'a', 'large', 'repository', '.', 'It', 'involves',
'searching', 'for', 'relevant', 'data', ',', 'documents', ',', 'or', 'other',
'resources', 'based', 'on', 'user', 'queries', '.', 'The', 'main', 'goal', 'of',
'information', 'retrieval', 'is', 'to', 'provide', 'users', 'with', 'accurate',
'and', 'relevant', 'information', 'quickly', '.', 'Search', 'engines', 'like',
'Google', 'are', 'popular', 'examples', 'of', 'information', 'retrieval',
'systems', '.', 'They', 'use', 'algorithms', 'to', 'index', 'and', 'rank',
'web', 'pages', 'based', 'on', 'various', 'factors', 'like', 'keywords', 'and',
'user', 'behavior', '.', 'Information', 'retrieval', 'is', 'a', 'critical',
'component', 'of', 'many', 'applications', ',', 'including', 'digital',
'libraries', ',', 'databases', ',', 'and', 'online', 'search', 'engines', '.',
'In', 'information', 'retrieval', ',', 'relevance', 'is', 'a', 'key', 'factor',
'in', 'determining', 'the', 'usefulness', 'of', 'the', 'retrieved',
'information', '.', 'There', 'are', 'various', 'models', 'of', 'information',
'retrieval', ',', 'including', 'the', 'Boolean', 'model', ',', 'the', 'vector',
'space', 'model', ',', 'and', 'probabilistic', 'models', '.', 'These', 'models',
'differ', 'in', 'how', 'they', 'represent', 'documents', 'and', 'queries', ',',
'and', 'how', 'they', 'measure', 'similarity', 'between', 'them', '.',
'Information', 'retrieval', 'systems', 'often', 'use', 'natural', 'language',
'processing', 'techniques', 'to', 'improve', 'search', 'accuracy', '.', 'Text',
'preprocessing', 'steps', 'like', 'tokenization', ',', 'stemming', ',', 'and',
'stop', 'word', 'removal', 'are', 'essential', 'in', 'information', 'retrieval',
'.', 'Machine', 'learning', 'is', 'increasingly', 'being', 'used', 'in',
'information', 'retrieval', 'to', 'enhance', 'search', 'results', 'and', 'user',
'experience', '.', 'Information', 'retrieval', 'also', 'involves', 'the',
'evaluation', 'of', 'system', 'performance', 'using', 'metrics', 'like',
'precision', ',', 'recall', ',', 'and', 'F1-score', '.', 'The', 'field', 'of',
'information', 'retrieval', 'is', 'constantly', 'evolving', 'with',
'advancements', 'in', 'technology', 'and', 'changes', 'in', 'user', 'behavior',
'.', 'Personalization', 'is', 'an', 'important', 'aspect', 'of', 'modern',
'information', 'retrieval', 'systems', ',', 'tailoring', 'results', 'to',
'individual', 'user', 'preferences', '.', 'Information', 'retrieval', 'plays',
'a', 'significant', 'role', 'in', 'e-commerce', ',', 'helping', 'users', 'find',
'products', 'and', 'services', 'online', '.', 'Social', 'media', 'platforms',
```

'also', 'rely', 'on', 'information', 'retrieval', 'to', 'surface', 'relevant',
'content', 'to', 'users', '.', 'Big', 'data', 'and', 'information', 'retrieval',
'are', 'closely', 'linked', ',', 'as', 'large', 'datasets', 'require',
'efficient', 'retrieval', 'mechanisms', '.', 'Information', 'retrieval', 'is',
'not', 'limited', 'to', 'text', ';', 'it', 'can', 'also', 'involve', 'images',
',', 'videos', ',', 'and', 'other', 'multimedia', 'content', '.', 'One', 'of',
'the', 'challenges', 'in', 'information', 'retrieval', 'is', 'handling',
'ambiguous', 'queries', 'that', 'may', 'have', 'multiple', 'interpretations',
'.', 'Information', 'retrieval', 'systems', 'must', 'also', 'consider', 'the',
'context', 'in', 'which', 'a', 'query', 'is', 'made', 'to', 'provide', 'more',
'relevant', 'results', '.', 'Privacy', 'and', 'security', 'are', 'important',
'concerns', 'in', 'information', 'retrieval', ',', 'especially', 'when',
'dealing', 'with', 'sensitive', 'information', '.', 'Advancements', 'in',
'artificial', 'intelligence', 'are', 'driving', 'the', 'development', 'of',
'more', 'sophisticated', 'information', 'retrieval', 'systems', '.', 'User',
'feedback', 'is', 'often', 'used', 'to', 'refine', 'and', 'improve',
'information', 'retrieval', 'algorithms', 'over', 'time', '.', 'The', 'user',
'interface', 'of', 'an', 'information', 'retrieval', 'system', 'can', 'greatly',
'impact', 'the', 'user', 'experience', 'and', 'satisfaction', '.',
'Information', 'retrieval', 'can', 'be', 'applied', 'to', 'various', 'domains',
',', 'including', 'healthcare', ',', 'finance', ',', 'and', 'education', '.',
'The', 'future', 'of', 'information', 'retrieval', 'may', 'involve', 'more',
'conversational', 'and', 'interactive', 'search', 'experiences', '.', 'Voice',
'search', 'is', 'an', 'emerging', 'trend', 'in', 'information', 'retrieval',
',', 'driven', 'by', 'the', 'popularity', 'of', 'virtual', 'assistants', 'like',
'Siri', 'and', 'Alexa', '.', 'Cross-lingual', 'information', 'retrieval',
'allows', 'users', 'to', 'search', 'for', 'information', 'in', 'one',
'language', 'and', 'retrieve', 'results', 'in', 'another', '.', 'Information',
'retrieval', 'techniques', 'are', 'also', 'used', 'in', 'sentiment', 'analysis',
'to', 'gauge', 'public', 'opinion', 'on', 'various', 'topics', '.', 'Effective',
'information', 'retrieval', 'requires', 'a', 'combination', 'of', 'technical',
'expertise', 'and', 'a', 'deep', 'understanding', 'of', 'user', 'needs', '.',
'The', 'success', 'of', 'an', 'information', 'retrieval', 'system', 'often',
'depends', 'on', 'the', 'quality', 'of', 'the', 'underlying', 'data', 'and',
'the', 'algorithms', 'used', '.', 'Open-source', 'information', 'retrieval',
'frameworks', 'like', 'Apache', 'Lucene', 'and', 'Elasticsearch', 'are',
'widely', 'used', 'in', 'industry', '.', 'Research', 'in', 'information',
'retrieval', 'continues', 'to', 'push', 'the', 'boundaries', 'of', 'what', 'is',
'possible', 'in', 'the', 'field', '.', 'The', 'integration', 'of',
'information', 'retrieval', 'with', 'other', 'technologies', ',', 'like',
'blockchain', ',', 'is', 'an', 'area', 'of', 'active', 'exploration', '.',
'Education', 'and', 'training', 'in', 'information', 'retrieval', 'are',
'important', 'for', 'developing', 'skilled', 'professionals', 'in', 'the',
'field', '.', 'Information', 'retrieval', 'systems', 'must', 'balance',
'between', 'speed', 'and', 'accuracy', 'to', 'meet', 'user', 'expectations',
'.', 'The', 'scalability', 'of', 'an', 'information', 'retrieval', 'system',
'is', 'crucial', 'for', 'handling', 'large', 'volumes', 'of', 'data', '.',
'Information', 'retrieval', 'is', 'a', 'fundamental', 'aspect', 'of',

```
'knowledge', 'management', 'and', 'information', 'systems', '.']

***tokens after removing stop_words: ***
 ['Information', 'retrieval', 'process', 'obtaining', 'information', 'large',
'repository', '.', 'involves', 'searching', 'relevant', 'data', ',',
'documents', ',', 'resources', 'based', 'user', 'queries', '.', 'main', 'goal',
'information', 'retrieval', 'provide', 'users', 'accurate', 'relevant',
'information', 'quickly', '.', 'Search', 'engines', 'like', 'Google', 'popular',
'examples', 'information', 'retrieval', 'systems', '.', 'use', 'algorithms',
'index', 'rank', 'web', 'pages', 'based', 'various', 'factors', 'like',
'keywords', 'user', 'behavior', '.', 'Information', 'retrieval', 'critical',
'component', 'many', 'applications', ',', 'including', 'digital', 'libraries',
',', 'databases', ',', 'online', 'search', 'engines', '.', 'information',
'retrieval', ',', 'relevance', 'key', 'factor', 'determining', 'usefulness',
'retrieved', 'information', '.', 'various', 'models', 'information',
'retrieval', ',', 'including', 'Boolean', 'model', ',', 'vector', 'space',
'model', ',', 'probabilistic', 'models', '.', 'models', 'differ', 'represent',
'documents', 'queries', ',', 'measure', 'similarity', '.', 'Information',
'retrieval', 'systems', 'often', 'use', 'natural', 'language', 'processing',
'techniques', 'improve', 'search', 'accuracy', '.', 'Text', 'preprocessing',
'steps', 'like', 'tokenization', ',', 'stemming', ',', 'stop', 'word',
'removal', 'essential', 'information', 'retrieval', '.', 'Machine', 'learning',
'increasingly', 'used', 'information', 'retrieval', 'enhance', 'search',
'results', 'user', 'experience', '.', 'Information', 'retrieval', 'also',
'involves', 'evaluation', 'system', 'performance', 'using', 'metrics', 'like',
'precision', ',', 'recall', ',', 'F1-score', '.', 'field', 'information',
'retrieval', 'constantly', 'evolving', 'advancements', 'technology', 'changes',
'user', 'behavior', '.', 'Personalization', 'important', 'aspect', 'modern',
'information', 'retrieval', 'systems', ',', 'tailoring', 'results',
'individual', 'user', 'preferences', '.', 'Information', 'retrieval', 'plays',
'significant', 'role', 'e-commerce', ',', 'helping', 'users', 'find',
'products', 'services', 'online', '.', 'Social', 'media', 'platforms', 'also',
'rely', 'information', 'retrieval', 'surface', 'relevant', 'content', 'users',
'.', 'Big', 'data', 'information', 'retrieval', 'closely', 'linked', ',',
'large', 'datasets', 'require', 'efficient', 'retrieval', 'mechanisms', '.',
'Information', 'retrieval', 'limited', 'text', ';', 'also', 'involve', 'images',
',', 'videos', ',', 'multimedia', 'content', '.', 'One', 'challenges',
'information', 'retrieval', 'handling', 'ambiguous', 'queries', 'may',
'multiple', 'interpretations', '.', 'Information', 'retrieval', 'systems',
'must', 'also', 'consider', 'context', 'query', 'made', 'provide', 'relevant',
'results', '.', 'Privacy', 'security', 'important', 'concerns', 'information',
'retrieval', ',', 'especially', 'dealing', 'sensitive', 'information', '.',
'Advancements', 'artificial', 'intelligence', 'driving', 'development',
'sophisticated', 'information', 'retrieval', 'systems', '.', 'User', 'feedback',
'often', 'used', 'refine', 'improve', 'information', 'retrieval', 'algorithms',
'time', '.', 'user', 'interface', 'information', 'retrieval', 'system',
'greatly', 'impact', 'user', 'experience', 'satisfaction', '.', 'Information',
'retrieval', 'applied', 'various', 'domains', ',', 'including', 'healthcare',
```

',', 'finance', ',', 'education', '.', 'future', 'information', 'retrieval',
'may', 'involve', 'conversational', 'interactive', 'search', 'experiences', '.',
'Voice', 'search', 'emerging', 'trend', 'information', 'retrieval', ',',
'driven', 'popularity', 'virtual', 'assistants', 'like', 'Siri', 'Alexa', '.',
'Cross-lingual', 'information', 'retrieval', 'allows', 'users', 'search',
'information', 'one', 'language', 'retrieve', 'results', 'another', '.',
'Information', 'retrieval', 'techniques', 'also', 'used', 'sentiment',
'analysis', 'gauge', 'public', 'opinion', 'various', 'topics', '.', 'Effective',
'information', 'retrieval', 'requires', 'combination', 'technical', 'expertise',
'deep', 'understanding', 'user', 'needs', '.', 'success', 'information',
'retrieval', 'system', 'often', 'depends', 'quality', 'underlying', 'data',
'algorithms', 'used', '.', 'Open-source', 'information', 'retrieval',
'frameworks', 'like', 'Apache', 'Lucene', 'Elasticsearch', 'widely', 'used',
'industry', '.', 'Research', 'information', 'retrieval', 'continues', 'push',
'boundaries', 'possible', 'field', '.', 'integration', 'information',
'retrieval', 'technologies', ',', 'like', 'blockchain', ',', 'area', 'active',
'exploration', '.', 'Education', 'training', 'information', 'retrieval',
'important', 'developing', 'skilled', 'professionals', 'field', '.',
'Information', 'retrieval', 'systems', 'must', 'balance', 'speed', 'accuracy',
'meet', 'user', 'expectations', '.', 'scalability', 'information', 'retrieval',
'system', 'crucial', 'handling', 'large', 'volumes', 'data', '.', 'Information',
'retrieval', 'fundamental', 'aspect', 'knowledge', 'management', 'information',
'systems', '.']

***Tokens after stemming: ***
 ['Information', 'retrieval', 'is', 'the', 'process', 'of', 'obtaining',
'information', 'from', 'a', 'large', 'repository', '.', 'It', 'involves',
'searching', 'for', 'relevant', 'data', ',', 'documents', ',', 'or', 'other',
'resources', 'based', 'on', 'user', 'queries', '.', 'The', 'main', 'goal', 'of',
'information', 'retrieval', 'is', 'to', 'provide', 'users', 'with', 'accurate',
'and', 'relevant', 'information', 'quickly', '.', 'Search', 'engines', 'like',
'Google', 'are', 'popular', 'examples', 'of', 'information', 'retrieval',
'systems', '.', 'They', 'use', 'algorithms', 'to', 'index', 'and', 'rank',
'web', 'pages', 'based', 'on', 'various', 'factors', 'like', 'keywords', 'and',
'user', 'behavior', '.', 'Information', 'retrieval', 'is', 'a', 'critical',
'component', 'of', 'many', 'applications', ',', 'including', 'digital',
'libraries', ',', 'databases', ',', 'and', 'online', 'search', 'engines', '.',
'In', 'information', 'retrieval', ',', 'relevance', 'is', 'a', 'key', 'factor',
'in', 'determining', 'the', 'usefulness', 'of', 'the', 'retrieved',
'information', '.', 'There', 'are', 'various', 'models', 'of', 'information',
'retrieval', ',', 'including', 'the', 'Boolean', 'model', ',', 'the', 'vector',
'space', 'model', ',', 'and', 'probabilistic', 'models', '.', 'These', 'models',
'differ', 'in', 'how', 'they', 'represent', 'documents', 'and', 'queries', ',',
'and', 'how', 'they', 'measure', 'similarity', 'between', 'them', '.',
'Information', 'retrieval', 'systems', 'often', 'use', 'natural', 'language',
'processing', 'techniques', 'to', 'improve', 'search', 'accuracy', '.', 'Text',
'preprocessing', 'steps', 'like', 'tokenization', ',', 'stemming', ',', 'and',
'stop', 'word', 'removal', 'are', 'essential', 'in', 'information', 'retrieval',

week3

October 14, 2024

[9]:
```
import re
from collections import defaultdict
```

[12]:
```
def tokenize(text):
    tokens = re.findall(r'\b\w+\b', text.lower())  # Find all word characters␣
 ↪(alphanumeric)
    return tokens

def create_inverted_index(documents):
    inverted_index = defaultdict(list)  # Dictionary to store terms and their␣
 ↪posting lists

    for doc_id, doc in documents.items():  # Iterate over the dictionary of␣
 ↪documents
        tokens = tokenize(doc)  # Tokenize the document
        unique_tokens = sorted(set(tokens))  # Remove duplicates and sort the␣
 ↪tokens

        for token in unique_tokens:
            inverted_index[token].append(doc_id)  # Append document ID to the␣
 ↪posting list

    return dict(inverted_index)

def display_inverted_index(inverted_index):
    print(f"{'Term':<15}{'Posting List'}")
    print('-' * 30)
    for term, posting_list in sorted(inverted_index.items()):
        print(f"{term:<15}{posting_list}")
```

[13]:
```
documents = {
    1 : "I did enact Julius Caesar: I was killed in the capitol, Brutus killed␣
 ↪me.",
    2 : "So let it be with Caesar. The noble Brutus hath told you Caesar was␣
 ↪ambitious:"
}
```

```
inverted_index = create_inverted_index(documents)

display_inverted_index(inverted_index)
```

```
Term           Posting List
------------------------------
ambitious      [2]
be             [2]
brutus         [1, 2]
caesar         [1, 2]
capitol        [1]
did            [1]
enact          [1]
hath           [2]
i              [1]
in             [1]
it             [2]
julius         [1]
killed         [1]
let            [2]
me             [1]
noble          [2]
so             [2]
the            [1, 2]
told           [2]
was            [1, 2]
with           [2]
you            [2]
```

[ ]:

AP21110010918_task1

November 7, 2024

[3]:
```
# document0 = "I did enact Julius Caesar I was killed i' the Capitol; Brutus␣
 ↪killed me."
document0 = "meat time traffic iron simple. depth poet task prompt cut.␣
 ↪strategy manufacturer cup gap working. cigarette price egg wonder cloud.␣
 ↪opportunity conclusion tour towel special. communication range sky upper␣
 ↪comment. driver name savings mess past. problem airport attempt package␣
 ↪tackle. performance restaurant animal rush concert. player obligation pain␣
 ↪rough pound. city company exchange hang purchase. boyfriend committee club␣
 ↪routine give. statement drawer figure extreme anything. cousin cause ground␣
 ↪sell final. wealth leader option partner crack. foundation body wave swim␣
 ↪public. method picture block stomach diet. excitement bonus lesson strain␣
 ↪load. area economy plastic sweet bite. setting research title upstairs boot.␣
 ↪leadership poem cycle mail human. university weather question habit produce.␣
 ↪army instruction edge ordinary spend. oven presence log emergency a.␣
 ↪guidance reputation suit advance visit. ability action culture smoke bite.␣
 ↪wood cause animal wait you. secretary school confidence estimate help.␣
 ↪energy assignment frame designer keep. efficiency boat mark winter working.␣
 ↪loss access band struggle primary. energy possibility copy young plane.␣
 ↪freedom airport traffic spring witness. hotel business pair press cable.␣
 ↪area boat bottom hit guy. health time vegetable cake plate. marriage wife␣
 ↪relative designer tackle. comparison gene stable somewhere truck. teacher␣
 ↪dad guarantee brilliant leave. player day brain chip engineer. problem␣
 ↪establishment reference pause scheme. boyfriend presentation credit curve␣
 ↪individual. ability analyst gift habit switch. internet heat land slide␣
 ↪tackle. apartment fish path anybody fall. recording practice discipline␣
 ↪surround camp. distribution page condition shake feel. tension girl metal␣
 ↪advance special. ad meal gift telephone shoe. preparation appointment party␣
 ↪dump ring."
document1 = "So let it be with Caesar. The noble Brutus hath told you Caesar␣
 ↪was ambitious."
documents = [document0, document1]
```

[4]:
```
# Query0 = "brutus AND caesar"
Query0 = "meal"
Query1 = "brutus AND julius AND caesar"
Query2 = "brutus AND julius AND NOT caesar"
queries = [Query0, Query1, Query2]
```

[5]:
```
def boolean_query(query, docs):
    query = query.lower()
    terms = query.split()

    result_set = set()

    current_op = ""

    for term in terms:
        if term == "AND":
            current_op = "AND"
        elif term == "OR":
            current_op = "OR"
        elif term == "NOT":
            current_op = "NOT"
        else:
            matching_docs = {i for i, doc in enumerate(docs) if term in doc.
 ↪lower()}

            if result_set == set():
                result_set = matching_docs
            else:
                if current_op == "AND":
                    result_set &= matching_docs
                elif current_op == "OR":
                    result_set |= matching_docs
                elif current_op == "NOT":
                    result_set -= matching_docs

    return [docs[i] for i in result_set]
```

[6]:
```
q = 1
for query in queries:
    d = 1
    for doc in documents:
        matching_documents = boolean_query(query, documents)
        print(f"for query{q} Matchings in document{d}:")
        for match in matching_documents:
            print(f"- {match}")
        d+=1
    q += 1
    print()
```

```
for query1 Matchings in document1:
- meat time traffic iron simple. depth poet task prompt cut. strategy
manufacturer cup gap working. cigarette price egg wonder cloud. opportunity
conclusion tour towel special. communication range sky upper comment. driver
name savings mess past. problem airport attempt package tackle. performance
```

restaurant animal rush concert. player obligation pain rough pound. city company exchange hang purchase. boyfriend committee club routine give. statement drawer figure extreme anything. cousin cause ground sell final. wealth leader option partner crack. foundation body wave swim public. method picture block stomach diet. excitement bonus lesson strain load. area economy plastic sweet bite. setting research title upstairs boot. leadership poem cycle mail human. university weather question habit produce. army instruction edge ordinary spend. oven presence log emergency a. guidance reputation suit advance visit. ability action culture smoke bite. wood cause animal wait you. secretary school confidence estimate help. energy assignment frame designer keep. efficiency boat mark winter working. loss access band struggle primary. energy possibility copy young plane. freedom airport traffic spring witness. hotel business pair press cable. area boat bottom hit guy. health time vegetable cake plate. marriage wife relative designer tackle. comparison gene stable somewhere truck. teacher dad guarantee brilliant leave. player day brain chip engineer. problem establishment reference pause scheme. boyfriend presentation credit curve individual. ability analyst gift habit switch. internet heat land slide tackle. apartment fish path anybody fall. recording practice discipline surround camp. distribution page condition shake feel. tension girl metal advance special. ad meal gift telephone shoe. preparation appointment party dump ring.
for query1 Matchings in document2:
- meat time traffic iron simple. depth poet task prompt cut. strategy manufacturer cup gap working. cigarette price egg wonder cloud. opportunity conclusion tour towel special. communication range sky upper comment. driver name savings mess past. problem airport attempt package tackle. performance restaurant animal rush concert. player obligation pain rough pound. city company exchange hang purchase. boyfriend committee club routine give. statement drawer figure extreme anything. cousin cause ground sell final. wealth leader option partner crack. foundation body wave swim public. method picture block stomach diet. excitement bonus lesson strain load. area economy plastic sweet bite. setting research title upstairs boot. leadership poem cycle mail human. university weather question habit produce. army instruction edge ordinary spend. oven presence log emergency a. guidance reputation suit advance visit. ability action culture smoke bite. wood cause animal wait you. secretary school confidence estimate help. energy assignment frame designer keep. efficiency boat mark winter working. loss access band struggle primary. energy possibility copy young plane. freedom airport traffic spring witness. hotel business pair press cable. area boat bottom hit guy. health time vegetable cake plate. marriage wife relative designer tackle. comparison gene stable somewhere truck. teacher dad guarantee brilliant leave. player day brain chip engineer. problem establishment reference pause scheme. boyfriend presentation credit curve individual. ability analyst gift habit switch. internet heat land slide tackle. apartment fish path anybody fall. recording practice discipline surround camp. distribution page condition shake feel. tension girl metal advance special. ad meal gift telephone shoe. preparation appointment party dump ring.

for query2 Matchings in document1:
- So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious.

```
for query2 Matchings in document2:
- So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious.

for query3 Matchings in document1:
- So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious.
for query3 Matchings in document2:
- So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious.
```

task2

November 7, 2024

[1]:
```
documents = []
for i in range(1,101):
    doc_path = "AP21110010918/document"+str(i)+".txt"
    f = open(doc_path,"r")
    doc = f.read()
    documents.append(doc)
```

[2]:
```
def boolean_query(query, docs):
    query = query.lower()
    terms = query.split()

    result_set = set()

    current_op = ""

    for term in terms:
        if term == "AND":
            current_op = "AND"
        elif term == "OR":
            current_op = "OR"
        elif term == "NOT":
            current_op = "NOT"
        else:
            matching_docs = {i for i, doc in enumerate(docs) if term in doc.
 ↪lower()}

            if result_set == set():
                result_set = matching_docs
            else:
                if current_op == "AND":
                    result_set &= matching_docs
                elif current_op == "OR":
                    result_set |= matching_docs
                elif current_op == "NOT":
                    result_set -= matching_docs

    return [docs[i] for i in result_set]
```

[3]:
```
queries = ["meal",
"potato",
"efficiency AND NOT son",
"emotion AND NOT answer",
"county AND NOT wealth AND conclusion",
"event OR NOT population OR NOT passenger",
"memory OR NOT organization AND preference OR NOT oil",
"bird OR NOT agreement OR owner AND contribution",
"person AND nature OR food OR responsibility AND NOT revenue",
"region AND NOT data AND NOT cousin AND NOT phone AND NOT air"]
```

[4]:
```
q = 1
for query in queries:
    d = 1
    for doc in documents:
        matching_documents = boolean_query(query, documents)
        print(f"for query{q} Matchings in document{d}:")
        for match in matching_documents:
            print(f"- {match}")
        d+=1
    q += 1
    print()
```

```
IOPub data rate exceeded.
The Jupyter server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--ServerApp.iopub_data_rate_limit`.

Current values:
ServerApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
ServerApp.rate_limit_window=3.0 (secs)


for query6 Matchings in document9:
- memory experience angle host give. media coast tour handle primary. football
focus advantage print sandwich. food possibility copy split candy.
administration gene baby raw sail. highway salad speed prior play. environment
hand window tourist fault. player platform shop race visit. height boss friend
estimate worth. medicine promotion substance brief teach. relationship record
friend extreme still. argument uncle relative knife cost. investment building
structure annual purple. population relation confidence bill conference.
communication ladder text grab official. customer type scale strain stroke.
event lab juice drag concern. instance web hole drink load. theory practice
catch hit plane. technology committee strength repair being. finding song
minimum twist promise. entry proposal skin mobile simple. marketing refrigerator
suit hate search. activity queen reference scratch she. paper accident progress
chemical read. sample hearing minimum employ specific. aspect presentation
```

AP21110010918_Task1

November 9, 2024

0.0.1 Task1: Implementation of External memory indexing - BSBI

[5]:
```
import re
from collections import defaultdict
```

[17]:
```
document0 = "I did enact Julius Caesar I was killed i' the Capitol; Brutus␣
 ↪killed me."
document1 = "So let it be with Caesar. The noble Brutus hath told you Caesar␣
 ↪was ambitious."
```

[26]:
```
#creating blocks
def preprocess(document):
    # Lowercase and remove special characters
    document = document.lower()
    document = re.sub(r'[^\w\s]', '', document)
    return document.split()

# Create dictionary and posting list for each block
def create_posting_list(block, block_id):
    dictionary = defaultdict(list)
    for index, word in enumerate(block):
        dictionary[word].append((block_id, index))
    return dict(dictionary)

# Merging
def merge_posting_lists(pl1, pl2):
    merged_postings = defaultdict(list)
    all_terms = set(pl1.keys()).union(set(pl2.keys()))

    for term in all_terms:
        if term in pl1:
            merged_postings[term].extend(pl1[term])
        if term in pl2:
            merged_postings[term].extend(pl2[term])

    return dict(merged_postings)

# Perform basic boolean query processing
```

```
def boolean_query(merged_postings, term):
    return merged_postings.get(term, [])
```

[21]:
```
block0 = preprocess(document0)
block1 = preprocess(document1)
```

[22]:
```
# Create posting lists
posting_list_0 = create_posting_list(block0, 0)
posting_list_1 = create_posting_list(block1, 1)

print("Block 0 Posting List:", posting_list_0)
print("Block 1 Posting List:", posting_list_1)
```

```
Block 0 Posting List: {'i': [(0, 0), (0, 5), (0, 8)], 'did': [(0, 1)], 'enact':
[(0, 2)], 'julius': [(0, 3)], 'caesar': [(0, 4)], 'was': [(0, 6)], 'killed':
[(0, 7), (0, 12)], 'the': [(0, 9)], 'capitol': [(0, 10)], 'brutus': [(0, 11)],
'me': [(0, 13)]}
Block 1 Posting List: {'so': [(1, 0)], 'let': [(1, 1)], 'it': [(1, 2)], 'be':
[(1, 3)], 'with': [(1, 4)], 'caesar': [(1, 5), (1, 12)], 'the': [(1, 6)],
'noble': [(1, 7)], 'brutus': [(1, 8)], 'hath': [(1, 9)], 'told': [(1, 10)],
'you': [(1, 11)], 'was': [(1, 13)], 'ambitious': [(1, 14)]}
```

[29]:
```
merged_postings = merge_posting_lists(posting_list_0, posting_list_1)
print("\nMerged Posting List:", merged_postings)
```

```
Merged Posting List: {'did': [(0, 1)], 'let': [(1, 1)], 'the': [(0, 9), (1, 6)],
'capitol': [(0, 10)], 'me': [(0, 13)], 'caesar': [(0, 4), (1, 5), (1, 12)],
'was': [(0, 6), (1, 13)], 'brutus': [(0, 11), (1, 8)], 'killed': [(0, 7), (0,
12)], 'you': [(1, 11)], 'be': [(1, 3)], 'with': [(1, 4)], 'enact': [(0, 2)],
'noble': [(1, 7)], 'ambitious': [(1, 14)], 'hath': [(1, 9)], 'i': [(0, 0), (0,
5), (0, 8)], 'it': [(1, 2)], 'julius': [(0, 3)], 'so': [(1, 0)], 'told': [(1,
10)]}

Boolean Query Result for 'caesar': [(0, 4), (1, 5), (1, 12)]
```

[30]:
```
# Search for the word 'caesar'
term = "caesar"
result = boolean_query(merged_postings, term)
print(f"\nBoolean Query Result for '{term}':", result)
```

```
Boolean Query Result for 'caesar': [(0, 4), (1, 5), (1, 12)]
```

AP21110010918_Task2

November 9, 2024

0.0.1 Task2: Implementation of External memory indexing - SPIMI

[4]:
```
from collections import defaultdict
```

[2]:
```
# Preprocess text and split into blocks and remove special Characters
def preprocess(document):
    document = document.lower()
    document = re.sub(r'[^\w\s]', '', document)
    return document.split()

# SPIMI invert (process each block, update dictionary)
def spimi_invert(block, block_id):
    dictionary = defaultdict(list)
    for index, term in enumerate(block):
        dictionary[term].append((block_id, index))
    return dict(dictionary)

# Merge two dictionaries
def merge_dictionaries(dict1, dict2):
    merged_dict = defaultdict(list)
    all_terms = set(dict1.keys()).union(set(dict2.keys()))

    for term in all_terms:
        if term in dict1:
            merged_dict[term].extend(dict1[term])
        if term in dict2:
            merged_dict[term].extend(dict2[term])

    return dict(merged_dict)
```

[7]:
```
document0 = "I did enact Julius Caesar I was killed i' the Capitol; Brutus␣
 ↪killed me."
document1 = "So let it be with Caesar. The noble Brutus hath told you Caesar␣
 ↪was ambitious."
```

[8]:
```
# Creating blocks
block0 = preprocess(document0)
block1 = preprocess(document1)
```

[10]:
```
# Process each block using SPIMI
spimi_dict_block0 = spimi_invert(block0, 0)
spimi_dict_block1 = spimi_invert(block1, 1)

print("SPIMI Block 0 Dictionary:", spimi_dict_block0)
print("SPIMI Block 1 Dictionary:", spimi_dict_block1)
```

```
SPIMI Block 0 Dictionary: {'i': [(0, 0), (0, 5), (0, 8)], 'did': [(0, 1)],
'enact': [(0, 2)], 'julius': [(0, 3)], 'caesar': [(0, 4)], 'was': [(0, 6)],
'killed': [(0, 7), (0, 12)], 'the': [(0, 9)], 'capitol': [(0, 10)], 'brutus':
[(0, 11)], 'me': [(0, 13)]}
SPIMI Block 1 Dictionary: {'so': [(1, 0)], 'let': [(1, 1)], 'it': [(1, 2)],
'be': [(1, 3)], 'with': [(1, 4)], 'caesar': [(1, 5), (1, 12)], 'the': [(1, 6)],
'noble': [(1, 7)], 'brutus': [(1, 8)], 'hath': [(1, 9)], 'told': [(1, 10)],
'you': [(1, 11)], 'was': [(1, 13)], 'ambitious': [(1, 14)]}
```

[11]:
```
# merge the files
merged_spimi_dict = merge_dictionaries(spimi_dict_block0, spimi_dict_block1)

print("\nMerged SPIMI Dictionary:", merged_spimi_dict)
```

```
Merged SPIMI Dictionary: {'did': [(0, 1)], 'was': [(0, 6), (1, 13)], 'killed':
[(0, 7), (0, 12)], 'brutus': [(0, 11), (1, 8)], 'hath': [(1, 9)], 'i': [(0, 0),
(0, 5), (0, 8)], 'ambitious': [(1, 14)], 'caesar': [(0, 4), (1, 5), (1, 12)],
'you': [(1, 11)], 'let': [(1, 1)], 'it': [(1, 2)], 'me': [(0, 13)], 'so': [(1,
0)], 'capitol': [(0, 10)], 'enact': [(0, 2)], 'the': [(0, 9), (1, 6)], 'with':
[(1, 4)], 'julius': [(0, 3)], 'noble': [(1, 7)], 'told': [(1, 10)], 'be': [(1,
3)]}
```

[14]:
```
# Perform basic boolean query processing
def boolean_query(merged_dict, term):
    return merged_dict.get(term, [])

# search for the word 'caesar'
query_term = "caesar"
result = boolean_query(merged_spimi_dict, query_term)
print(f"\nBoolean Query Result for '{query_term}':", result)
```

```
Boolean Query Result for 'caesar': [(0, 4), (1, 5), (1, 12)]
```

Dictionary compression

November 9, 2024

[1]:
```
import itertools
```

[2]:
```
def create_dictionary(block):
    """
    Create a dictionary mapping for a given block.
    Dictionary compression: each unique symbol gets a unique integer ID.
    """
    unique_symbols = list(set(block))
    symbol_to_id = {symbol: idx for idx, symbol in enumerate(unique_symbols)}
    id_to_symbol = {idx: symbol for idx, symbol in enumerate(unique_symbols)}
    return symbol_to_id, id_to_symbol

def compress_block(block, symbol_to_id):
    """
    Compress a block using the provided dictionary.
    """
    compressed_block = [symbol_to_id[symbol] for symbol in block]
    return compressed_block

def decompress_block(compressed_block, id_to_symbol):
    """
    Decompress a block using the provided dictionary.
    """
    decompressed_block = [id_to_symbol[idx] for idx in compressed_block]
    return decompressed_block

def compress_data(data, block_size):
    """
    Compress the entire data by dividing it into blocks and compressing each␣
 ↪block.
    """
    blocks = [data[i:i + block_size] for i in range(0, len(data), block_size)]
    compressed_data = []
    dictionaries = []

    for block in blocks:
        symbol_to_id, id_to_symbol = create_dictionary(block)
```

```
        compressed_block = compress_block(block, symbol_to_id)
        compressed_data.append(compressed_block)
        dictionaries.append(id_to_symbol)  # Save dictionary for decompression

    return compressed_data, dictionaries

def decompress_data(compressed_data, dictionaries):
    """
    Decompress the entire data by using the saved dictionaries for each block.
    """
    decompressed_data = []
    for compressed_block, id_to_symbol in zip(compressed_data, dictionaries):
        decompressed_block = decompress_block(compressed_block, id_to_symbol)
        decompressed_data.extend(decompressed_block)

    return ''.join(decompressed_data)

# Example Usage
if __name__ == "__main__":
    # Input data (for example, a string of repeating patterns)
    data = "aaabbcddddddeeeeaaaaabbbbccccc"

    # Define block size for blocking
    block_size = 6

    # Compress the data
    compressed_data, dictionaries = compress_data(data, block_size)

    print("Original Data:", data)
    print("Compressed Data:", compressed_data)
    print("Dictionaries:", dictionaries)

    # Decompress the data
    decompressed_data = decompress_data(compressed_data, dictionaries)
    print("Decompressed Data:", decompressed_data)

    # Check if the decompressed data matches the original
    assert data == decompressed_data, "Decompression failed!"
```

```
Original Data: aaabbcddddddeeeeaaaaabbbbccccc
Compressed Data: [[1, 1, 1, 2, 2, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 1],
[0, 0, 0, 1, 1, 1], [1, 0, 0, 0, 0, 0]]
Dictionaries: [{0: 'c', 1: 'a', 2: 'b'}, {0: 'd'}, {0: 'e', 1: 'a'}, {0: 'a', 1:
'b'}, {0: 'c', 1: 'b'}]
Decompressed Data: aaabbcddddddeeeeaaaaabbbbccccc
```

Logarithmic merge

November 9, 2024

[1]:
```
import heapq
from collections import defaultdict
```

[2]:
```
index_levels = []
```

[3]:
```
def add_to_index(doc_id, terms):
    single_index = defaultdict(list)
    for term in terms:
        single_index[term].append(doc_id)

    # Add to index levels
    index_levels.append(single_index)
    logarithmic_merge()

def logarithmic_merge():
    while len(index_levels) > 1 and len(index_levels[-1]) ==␣
 ↪len(index_levels[-2]):
        merged_index = merge_indices(index_levels.pop(), index_levels.pop())
        index_levels.append(merged_index)

def merge_indices(index1, index2):
    # Merging two dictionaries of lists
    merged_index = defaultdict(list)
    for key in set(index1) | set(index2):
        merged_index[key] = list(heapq.merge(index1.get(key, []), index2.
 ↪get(key, [])))
    return merged_index
```

[4]:
```
add_to_index(1, ['hello', 'world'])
add_to_index(2, ['hello', 'python'])
add_to_index(3, ['grape', 'fig'])
add_to_index(4, ['fig', 'banana'])
add_to_index(5, ['grape', 'apple'])
add_to_index(6, ['banana', 'apple'])

for level in index_levels:
    print(dict(level))
```

```
{'python': [2], 'grape': [3], 'hello': [1, 2], 'banana': [4], 'world': [1],
'fig': [3, 4]}
{'grape': [5], 'apple': [5, 6], 'banana': [6]}
```

Posting Compression - Gamma codes

November 9, 2024

[12]:
```python
def gamma_encode(number):
    if number == 0:
        return ''
    binary_rep = bin(number)[2:]
    offset = binary_rep[1:]
    length = len(offset) * '1' + '0'
    return length + offset

def gamma_decode(encoded_str):
    i = 0
    decoded = []

    while i < len(encoded_str):
        # Read unary code
        length = 0
        while encoded_str[i] == '1':
            length += 1
            i += 1
        i += 1

        # Read offset of given length
        offset = encoded_str[i:i + length]
        number = int('1' + offset, 2)  # Recreate the original number
        decoded.append(number)
        i += length

    return decoded
```

[13]:
```python
numbers = [10, 15, 20]
encoded_list = [gamma_encode(num) for num in numbers]
print("Gamma Encoded:", encoded_list)

decoded_list = [gamma_decode(code) for code in encoded_list]
print("Gamma Decoded:", decoded_list)
```

```
Gamma Encoded: ['1110010', '1110111', '111100100']
Gamma Decoded: [[10], [15], [20]]
```

Dictionary compression

November 9, 2024

[3]:
```
import itertools
```

[4]:
```
def create_dictionary(block):
    """
    Create a dictionary mapping for a given block.
    Dictionary compression: each unique symbol gets a unique integer ID.
    """
    unique_symbols = list(set(block))
    symbol_to_id = {symbol: idx for idx, symbol in enumerate(unique_symbols)}
    id_to_symbol = {idx: symbol for idx, symbol in enumerate(unique_symbols)}
    return symbol_to_id, id_to_symbol

def compress_block(block, symbol_to_id):
    """
    Compress a block using the provided dictionary.
    """
    compressed_block = [symbol_to_id[symbol] for symbol in block]
    return compressed_block

def decompress_block(compressed_block, id_to_symbol):
    """
    Decompress a block using the provided dictionary.
    """
    decompressed_block = [id_to_symbol[idx] for idx in compressed_block]
    return decompressed_block

def compress_data(data, block_size):
    """
    Compress the entire data by dividing it into blocks and compressing each␣
 ↪block.
    """
    blocks = [data[i:i + block_size] for i in range(0, len(data), block_size)]
    compressed_data = []
    dictionaries = []

    for block in blocks:
        symbol_to_id, id_to_symbol = create_dictionary(block)
```

```
        compressed_block = compress_block(block, symbol_to_id)
        compressed_data.append(compressed_block)
        dictionaries.append(id_to_symbol)  # Save dictionary for decompression

    return compressed_data, dictionaries

def decompress_data(compressed_data, dictionaries):
    """
    Decompress the entire data by using the saved dictionaries for each block.
    """
    decompressed_data = []
    for compressed_block, id_to_symbol in zip(compressed_data, dictionaries):
        decompressed_block = decompress_block(compressed_block, id_to_symbol)
        decompressed_data.extend(decompressed_block)

    return ''.join(decompressed_data)

# Example Usage
if __name__ == "__main__":
    # Input data (for example, a string of repeating patterns)
    data = "aaabbcddddddeeeeaaaaabbbbccccc"

    # Define block size for blocking
    block_size = 6

    # Compress the data
    compressed_data, dictionaries = compress_data(data, block_size)

    print("Original Data:", data)
    print("Compressed Data:", compressed_data)
    print("Dictionaries:", dictionaries)

    # Decompress the data
    decompressed_data = decompress_data(compressed_data, dictionaries)
    print("Decompressed Data:", decompressed_data)

    # Check if the decompressed data matches the original
    assert data == decompressed_data, "Decompression failed!"
```

```
Original Data: aaabbcddddddeeeeaaaaabbbbccccc
Compressed Data: [[0, 0, 0, 2, 2, 1], [0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 0, 0],
[0, 0, 0, 1, 1, 1], [1, 0, 0, 0, 0, 0]]
Dictionaries: [{0: 'a', 1: 'c', 2: 'b'}, {0: 'd'}, {0: 'a', 1: 'e'}, {0: 'a', 1:
'b'}, {0: 'c', 1: 'b'}]
Decompressed Data: aaabbcddddddeeeeaaaaabbbbccccc
```

Ranked retrieval

November 9, 2024

[8]:
```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
import math
```

[4]:
```
def calculate_cosine_similarity(tfidf_matrix):
    return cosine_similarity(tfidf_matrix)
```

[6]:
```
def calculate_tfidf(corpus):
    """
    Calculate the TF-IDF matrix for a given corpus.
    """
    vectorizer = TfidfVectorizer()
    X = vectorizer.fit_transform(corpus)
    feature_names = vectorizer.get_feature_names_out()
    tfidf_matrix = X.toarray()

    return tfidf_matrix, feature_names
```

[9]:
```
corpus = [
    "the quick brown fox jumped over the lazy dog",
    "the dog lay on the grass",
    "the quick brown fox was very quick"
]

tfidf_matrix, feature_names = calculate_tfidf(corpus)

similarity_matrix = calculate_cosine_similarity(tfidf_matrix)

print("Cosine Similarity Matrix:")
print(similarity_matrix)
```

```
Cosine Similarity Matrix:
[[1.         0.34167802 0.48193531]
 [0.34167802 1.         0.12967846]
 [0.48193531 0.12967846 1.        ]]
```

EXP 2 - Ranked retrieval

October 21, 2024

0.0.1 Exp. 2: Ranked retrieval - Implementation of TF-IDF, Vector space model, Cosine similarity.

0.0.2 Using custom functions

[6]:
```
import numpy as np
import math
```

[7]:
```
documents = [
    "The sky is blue",
    "The sun is bright",
    "The sun in the blue sky is bright",
    "We can see the shining sun, the bright sun"
]
```

[8]:
```
docs = [doc.lower().split() for doc in documents]

vocabulary = sorted(set(word for doc in docs for word in doc))
print("Vocabulary:", vocabulary)

incident_matrix = np.zeros((len(docs), len(vocabulary)))

for i, doc in enumerate(docs):
    for word in doc:
        if word in vocabulary:
            incident_matrix[i][vocabulary.index(word)] = 1

print("\nIncident Matrix (Binary Term Frequency):")
print(incident_matrix)
```

```
Vocabulary: ['blue', 'bright', 'can', 'in', 'is', 'see', 'shining', 'sky',
'sun', 'sun,', 'the', 'we']

Incident Matrix (Binary Term Frequency):
[[1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0.]
 [0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0.]
 [1. 1. 0. 1. 1. 0. 0. 1. 1. 0. 1. 0.]
 [0. 1. 1. 0. 0. 1. 1. 0. 1. 1. 1. 1.]]
```

[9]:
```
tf_matrix = np.zeros((len(docs), len(vocabulary)))

for i, doc in enumerate(docs):
    for word in doc:
        tf_matrix[i][vocabulary.index(word)] += 1

print("\nTerm Frequency (TF) Matrix:")
print(tf_matrix)
```

```
Term Frequency (TF) Matrix:
[[1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0.]
 [0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0.]
 [1. 1. 0. 1. 1. 0. 0. 1. 1. 0. 2. 0.]
 [0. 1. 1. 0. 0. 1. 1. 0. 1. 1. 2. 1.]]
```

[10]:
```
# Calculate Inverse Document Frequency (IDF)
idf_matrix = np.zeros(len(vocabulary))

for j, term in enumerate(vocabulary):
    idf_matrix[j] = math.log(len(docs) / sum(incident_matrix[:, j]))

print("\nInverse Document Frequency (IDF) Matrix:")
print(idf_matrix)
```

```
Inverse Document Frequency (IDF) Matrix:
[0.69314718 0.28768207 1.38629436 1.38629436 0.28768207 1.38629436
 1.38629436 0.69314718 0.28768207 1.38629436 0.         1.38629436]
```

[11]:
```
# Calculate TF-IDF
tf_idf_matrix = np.zeros((len(docs), len(vocabulary)))

for i in range(len(docs)):
    for j in range(len(vocabulary)):
        tf_idf_matrix[i][j] = tf_matrix[i][j] * idf_matrix[j]

print("\nTF-IDF Matrix:")
print(tf_idf_matrix)
```

```
TF-IDF Matrix:
[[0.69314718 0.         0.         0.         0.28768207 0.
  0.         0.69314718 0.         0.         0.         0.        ]
 [0.         0.28768207 0.         0.         0.28768207 0.
  0.         0.         0.28768207 0.         0.         0.        ]
 [0.69314718 0.28768207 0.         1.38629436 0.28768207 0.
  0.         0.69314718 0.28768207 0.         0.         0.        ]
 [0.         0.28768207 1.38629436 0.         0.         1.38629436
```

```
 1.38629436 0.         0.28768207 1.38629436 0.         1.38629436]]
```

[12]:
```
# Cosine similarity function
def cosine_similarity(vec1, vec2):
    dot_product = np.dot(vec1, vec2)
    magnitude = np.linalg.norm(vec1) * np.linalg.norm(vec2)
    if not magnitude:
        return 0
    return dot_product / magnitude

# Calculate cosine similarity between documents
cosine_sim_matrix = np.zeros((len(docs), len(docs)))

for i in range(len(docs)):
    for j in range(len(docs)):
        cosine_sim_matrix[i][j] = cosine_similarity(tf_idf_matrix[i],␣
 ↪tf_idf_matrix[j])

print("\nCosine Similarity Matrix:")
print(cosine_sim_matrix)
```

```
Cosine Similarity Matrix:
[[1.         0.16258153 0.57735027 0.        ]
 [0.16258153 1.         0.28159946 0.10625101]
 [0.57735027 0.28159946 1.         0.02992023]
 [0.         0.10625101 0.02992023 1.        ]]
```

0.0.3 Using Inbuilt Functions

[13]:
```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

[15]:
```
tfidf_vectorizer = TfidfVectorizer()

tfidf_matrix = tfidf_vectorizer.fit_transform(documents)

print("Feature Names:", tfidf_vectorizer.get_feature_names_out())

cosine_sim_matrix = cosine_similarity(tfidf_matrix)

print("\nTF-IDF Matrix:\n", tfidf_matrix.toarray())
```

```
Feature Names: ['blue' 'bright' 'can' 'in' 'is' 'see' 'shining' 'sky' 'sun'
'the' 'we']

TF-IDF Matrix:
 [[0.56855566 0.         0.         0.         0.46029481 0.
```

```
 0.         0.56855566 0.         0.37632116 0.        ]
[0.         0.52210862 0.         0.         0.52210862 0.
 0.         0.         0.52210862 0.42685801 0.        ]
[0.36942264 0.29907946 0.         0.46856577 0.29907946 0.
 0.         0.36942264 0.29907946 0.48903412 0.        ]
[0.         0.23910199 0.37459947 0.         0.         0.37459947
 0.37459947 0.         0.47820398 0.39096309 0.37459947]]
```

[16]:
```
print("\nCosine Similarity Matrix:\n", cosine_sim_matrix)
```

```
Cosine Similarity Matrix:
 [[1.         0.40095959 0.74177327 0.14712768]
 [0.40095959 1.         0.67720403 0.54139736]
 [0.74177327 0.67720403 1.         0.40572577]
 [0.14712768 0.54139736 0.40572577 1.        ]]
```

Exp 3 - Ranked retrieval

November 10, 2024

0.0.1 Exp. 3: Ranked retrieval - Implementation of Binary Independence Model, Okapi BM25.

0.0.2 Using Custom Functions

[4]:
```
import numpy as np
import math
```

[1]:
```
documents = [
    "The sky is blue",
    "The sun is bright",
    "The sun in the blue sky is bright",
    "We can see the shining sun, the bright sun"
]
query = "bright sun".split()
```

[2]:
```
docs = [doc.lower().split() for doc in documents]
vocabulary = sorted(set(word for doc in docs for word in doc))
print("Vocabulary:", vocabulary)
```

```
Vocabulary: ['blue', 'bright', 'can', 'in', 'is', 'see', 'shining', 'sky',
'sun', 'sun,', 'the', 'we']
```

[5]:
```
incident_matrix = np.zeros((len(docs), len(vocabulary)))
for i, doc in enumerate(docs):
    for word in doc:
        if word in vocabulary:
            incident_matrix[i][vocabulary.index(word)] = 1
print("\nIncident Matrix (Binary Term Frequency):")
print(incident_matrix)
```

```
Incident Matrix (Binary Term Frequency):
[[1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0.]
 [0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0.]
 [1. 1. 0. 1. 1. 0. 0. 1. 1. 0. 1. 0.]
 [0. 1. 1. 0. 0. 1. 1. 0. 1. 1. 1. 1.]]
```

[6]:
```
query_vec = [1 if word in query else 0 for word in vocabulary]
scores_bim = np.dot(incident_matrix, query_vec)
print("\nRanked documents (BIM):")
for i in np.argsort(-scores_bim):
    print(f"Document {i+1} - Score: {scores_bim[i]}")
```

```
Ranked documents (BIM):
Document 2 - Score: 2.0
Document 3 - Score: 2.0
Document 4 - Score: 2.0
Document 1 - Score: 0.0
```

[7]:
```
k1 = 1.5
b = 0.75
doc_lengths = [len(doc) for doc in docs]
avg_doc_length = np.mean(doc_lengths)

# Function to calculate BM25 score
def bm25_score(doc, query, doc_len, avg_doc_len, k1=1.5, b=0.75):
    score = 0
    for term in query:
        if term in doc:
            tf = doc.count(term)
            df = sum(1 for d in docs if term in d)
            idf = math.log((len(docs) - df + 0.5) / (df + 0.5) + 1)
            numerator = tf * (k1 + 1)
            denominator = tf + k1 * (1 - b + b * (doc_len / avg_doc_len))
            score += idf * (numerator / denominator)
    return score
# Calculate BM25 scores for each document
bm25_scores = [bm25_score(doc, query, len(doc), avg_doc_length, k1, b) for doc␣
 ↪in docs]

# Display BM25 results
print("\nRanked documents (BM25):")
for i in np.argsort(bm25_scores):
    print(f"Document {i+1} - Score: {bm25_scores[i]}")
```

```
Ranked documents (BM25):
Document 1 - Score: 0
Document 4 - Score: 0.5954506576606551
Document 3 - Score: 0.6335256553085833
Document 2 - Score: 0.8512528494957815
```

0.0.3 Using inbuilt functions

[10]:
```
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
from rank_bm25 import BM25Okapi
```

[12]:
```
query = "bright sun"
vectorizer = CountVectorizer(binary=True)
X = vectorizer.fit_transform(documents).toarray()
query_vec = vectorizer.transform([query]).toarray()[0]

scores = np.dot(X, query_vec)

doc_indices = np.argsort(-scores)  # Sort in descending order of scores
print("Ranked documents (BIM):")
for i in doc_indices:
    print(f"Document {i+1} - Score: {scores[i]}")
```

```
Ranked documents (BIM):
Document 2 - Score: 2
Document 3 - Score: 2
Document 4 - Score: 2
Document 1 - Score: 0
```

[11]:
```
tokenized_docs = [doc.split() for doc in documents]
bm25 = BM25Okapi(tokenized_docs)
query = "bright sun".split()
bm25_scores = bm25.get_scores(query)
doc_indices = np.argsort(-bm25_scores)
print("\nRanked documents (BM25):")
for i in doc_indices:
    print(f"Document {i+1} - Score: {bm25_scores[i]}")
```

```
Ranked documents (BM25):
Document 2 - Score: 0.07777656144549326
Document 3 - Score: 0.05788344448607757
Document 4 - Score: 0.054404639809117986
Document 1 - Score: 0.0
```

Text-Document classification algorithms

October 28, 2024

0.0.1 Exp. 4: Implementation of Text/Document classification algorithms - Naive Bayes models, Rocchio classification, k-Nearest Neighbors, Support vector machine classifiers, Decision trees, Bagging, Boosting.

[1]:
```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier
from sklearn.metrics import classification_report, accuracy_score
```

[2]:
```
newsgroups = fetch_20newsgroups(subset='all', categories=['alt.atheism', 'sci.
 ↪space'])
X, y = newsgroups.data, newsgroups.target

vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

[7]:
```
classifiers = {
    "Naive Bayes": MultinomialNB(),
    "k-Nearest Neighbors": KNeighborsClassifier(n_neighbors=3),
    "Support Vector Machine": SVC(kernel='linear', probability=True),
    "Decision Tree": DecisionTreeClassifier(),
    "Bagging (with Decision Tree)":␣
 ↪BaggingClassifier(estimator=DecisionTreeClassifier(), n_estimators=10,␣
 ↪random_state=42), # Changed 'base_estimator' to 'estimator'
    "Boosting (AdaBoost with Decision Tree)":␣
 ↪AdaBoostClassifier(estimator=DecisionTreeClassifier(), n_estimators=50,␣
 ↪random_state=42) # Changed 'base_estimator' to 'estimator'
}
```

[8]:
```
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"\nClassifier: {name}")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(classification_report(y_test, y_pred, target_names=newsgroups.
 ↪target_names))
```

```
Classifier: Naive Bayes
Accuracy: 0.9963
              precision    recall  f1-score   support

 alt.atheism       1.00      0.99      1.00       237
   sci.space       0.99      1.00      1.00       299

    accuracy                           1.00       536
   macro avg       1.00      1.00      1.00       536
weighted avg       1.00      1.00      1.00       536


Classifier: k-Nearest Neighbors
Accuracy: 0.9944
              precision    recall  f1-score   support

 alt.atheism       1.00      0.99      0.99       237
   sci.space       0.99      1.00      0.99       299

    accuracy                           0.99       536
   macro avg       0.99      0.99      0.99       536
weighted avg       0.99      0.99      0.99       536


Classifier: Support Vector Machine
Accuracy: 0.9963
              precision    recall  f1-score   support

 alt.atheism       1.00      0.99      1.00       237
   sci.space       0.99      1.00      1.00       299

    accuracy                           1.00       536
   macro avg       1.00      1.00      1.00       536
weighted avg       1.00      1.00      1.00       536


Classifier: Decision Tree
Accuracy: 0.9366
              precision    recall  f1-score   support
```

```
 alt.atheism       0.94      0.92      0.93       237
   sci.space       0.93      0.95      0.94       299

    accuracy                           0.94       536
   macro avg       0.94      0.93      0.94       536
weighted avg       0.94      0.94      0.94       536


Classifier: Bagging (with Decision Tree)
Accuracy: 0.9534
              precision    recall  f1-score   support

 alt.atheism       0.96      0.93      0.95       237
   sci.space       0.94      0.97      0.96       299

    accuracy                           0.95       536
   macro avg       0.95      0.95      0.95       536
weighted avg       0.95      0.95      0.95       536


/usr/local/lib/python3.10/dist-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(


Classifier: Boosting (AdaBoost with Decision Tree)
Accuracy: 0.9310
              precision    recall  f1-score   support

 alt.atheism       0.94      0.90      0.92       237
   sci.space       0.92      0.96      0.94       299

    accuracy                           0.93       536
   macro avg       0.93      0.93      0.93       536
weighted avg       0.93      0.93      0.93       536
```

[9]:
```python
import numpy as np
```

[10]:
```python
n_examples = 5

for name, clf in classifiers.items():
    y_pred = clf.predict(X_test)

    print(f"\nClassifier: {name}")
```

```
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(classification_report(y_test, y_pred, target_names=newsgroups.
↪target_names))

    print("\nExample Predictions:")
    example_indices = np.random.choice(len(y_test), n_examples, replace=False)
    for i in example_indices:
        print(f"\nDocument: {newsgroups.data[i][:200]}...")
        print(f"Actual Label: {newsgroups.target_names[y_test[i]]}")
        print(f"Predicted Label: {newsgroups.target_names[y_pred[i]]}")
```

```
Classifier: Naive Bayes
Accuracy: 0.9963
              precision    recall  f1-score   support

 alt.atheism       1.00      0.99      1.00       237
   sci.space       0.99      1.00      1.00       299

    accuracy                           1.00       536
   macro avg       1.00      1.00      1.00       536
weighted avg       1.00      1.00      1.00       536


Example Predictions:

Document: From: mccall@mksol.dseg.ti.com (fred j mccall 575-3539)
Subject: Re: Vandalizing the sky.
Article-I.D.: mksol.1993Apr22.204742.10671
Organization: Texas Instruments Inc
Lines: 62

In <C5tvL2.1In@herme…
Actual Label: alt.atheism
Predicted Label: alt.atheism

Document: From: henry@zoo.toronto.edu (Henry Spencer)
Subject: Re: Boom! Whoosh…
Organization: U of Toronto Zoology
Lines: 21

In article <1r46ofINNdku@gap.caltech.edu> palmer@cco.caltech.edu (David M.
Pal…
Actual Label: alt.atheism
Predicted Label: alt.atheism

Document: From: baalke@kelvin.jpl.nasa.gov (Ron Baalke)
Subject: Successful Balloon Flight Measures Ozone Layer
```

```
Organization: Siemens-Nixdorf AG
Lines: 84
NNTP-Posting-Host: d012s658.ap.mchp.sni.de

In article <16…
Actual Label: alt.atheism
Predicted Label: alt.atheism
```

[11]:
```python
from sklearn.preprocessing import normalize
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

class RocchioClassifier:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.centroids = {}

        for cls in self.classes:
            cls_indices = np.where(y == cls)
            cls_samples = X[cls_indices]
            self.centroids[cls] = cls_samples.mean(axis=0).A

        self.centroids = {cls: normalize(centroid) for cls, centroid in self.
 ↪centroids.items()}

    def predict(self, X):
        predictions = []
        for x in X:
            x = normalize(x)
            similarities = {cls: cosine_similarity(x, centroid)[0, 0] for cls,␣
 ↪centroid in self.centroids.items()}
            predicted_class = max(similarities, key=similarities.get)
            predictions.append(predicted_class)
        return np.array(predictions)
rocchio_clf = RocchioClassifier()
rocchio_clf.fit(X_train, y_train)
rocchio_predictions = rocchio_clf.predict(X_test)

print("\nClassifier: Rocchio")
print(f"Accuracy: {accuracy_score(y_test, rocchio_predictions):.4f}")
print(classification_report(y_test, rocchio_predictions,␣
 ↪target_names=newsgroups.target_names))

n_examples = 5
print("\nExample Predictions:")
example_indices = np.random.choice(len(y_test), n_examples, replace=False)
for i in example_indices:
```

```
    print(f"\nDocument: {newsgroups.data[i][:200]}...")
    print(f"Actual Label: {newsgroups.target_names[y_test[i]]}")
    print(f"Predicted Label: {newsgroups.target_names[rocchio_predictions[i]]}")
```

```
Classifier: Rocchio
Accuracy: 0.9944
              precision    recall  f1-score   support

 alt.atheism       1.00      0.99      0.99       237
   sci.space       0.99      1.00      0.99       299

    accuracy                           0.99       536
   macro avg       0.99      0.99      0.99       536
weighted avg       0.99      0.99      0.99       536


Example Predictions:

Document: From: dennisn@ecs.comm.mot.com (Dennis Newkirk)
Subject: Re: Proton/Centaur?
Organization: Motorola
Nntp-Posting-Host: 145.1.146.43
Lines: 37

In article <1993Apr20.211638.168730@zeus.calpoly.edu> jgr…
Actual Label: alt.atheism
Predicted Label: alt.atheism

Document: From: geoff@poori.East.Sun.COM (Geoff Arnold @ Sun BOS - R.H. coast
near the top)
Subject: Re: Who has read Rushdie's _The Satanic Ve
Organization: SunSelect
Lines: 15
Distribution: world
Reply-To: ge…
Actual Label: alt.atheism
Predicted Label: alt.atheism

Document: From: qpliu@ernie.Princeton.EDU (q.p.liu)
Subject: Re: A visit from the Jehovah's Witnesses
Originator: news@nimaster
Nntp-Posting-Host: ernie.princeton.edu
Reply-To: qpliu@princeton.edu
Organization:…
Actual Label: sci.space
Predicted Label: sci.space
```

Document: From: wilsonr@logica.co.uk
Subject: Re: What it means to be human? (Was: PARSIFAL)
Distribution: world
Lines: 40

In article <1993Apr16.001326.15820@cs.ucla.edu>, Brad Pierce
<pierce@cs.ucla.edu> writ…
Actual Label: sci.space
Predicted Label: sci.space

Document: From: suopanki@stekt6.oulu.fi (Heikki T. Suopanki)
Subject: Re: A visit from the Jehovah's Witnesses
In-Reply-To: jbrown@batman.bmd.trw.com's message of 5 Apr 93 11:24:30 MST
Lines: 17
Reply-To: suopa…
Actual Label: sci.space
Predicted Label: sci.space

week10 - exp5 - TextDocument clustering

November 4, 2024

[1]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans, AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score
import nltk
from nltk.corpus import stopwords
import re
import numpy as np
```

[3]:
```
import os

# Directory containing text files
path = "E:\Semesters\SEM VII\Information Retreival\week10\dataset\sport"
file_contents = []

# Loop through all files in the directory
for filename in os.listdir(path):
    if filename.endswith(".txt"):
        file_path = os.path.join(path, filename)
        with open(file_path, 'r', encoding='utf-8') as file:
            content = file.read()
            file_contents.append(content)

print(f"Loaded {len(file_contents)} documents.")
```

```
Loaded 100 documents.
```

0.0.1 Preprocessing the text data

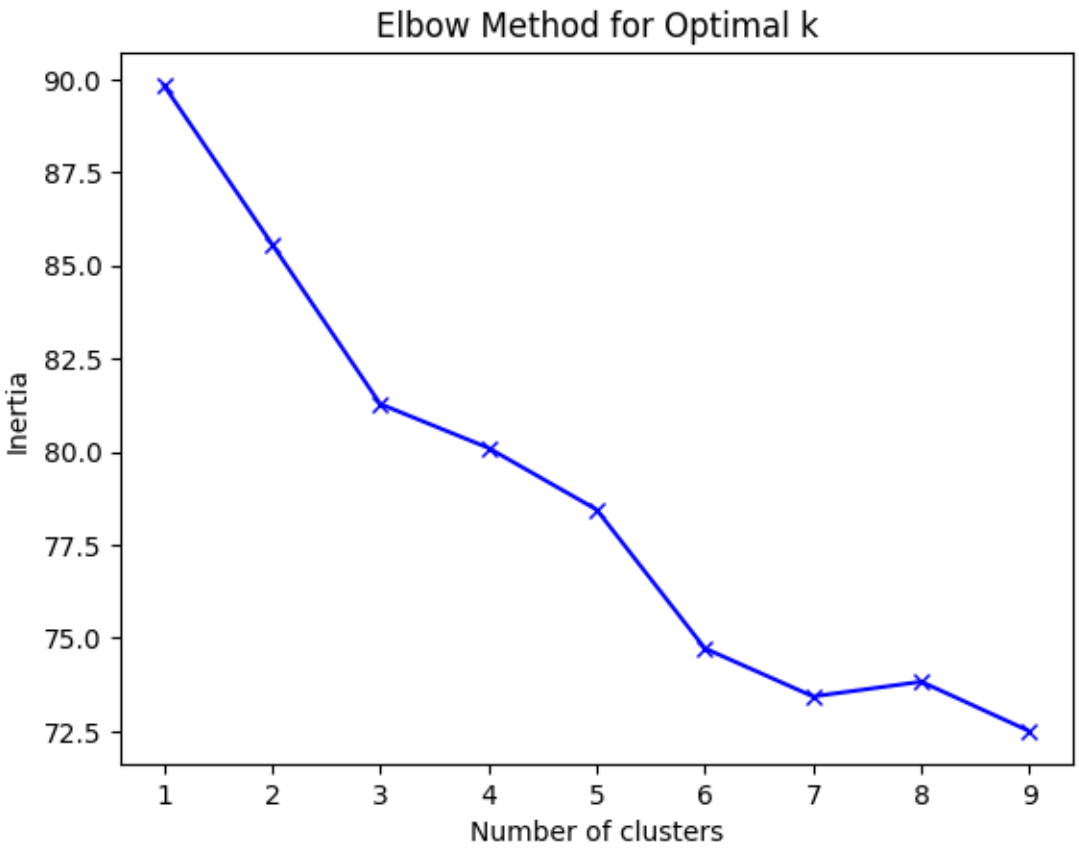
[4]:
```
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

def preprocess(text):
    text = re.sub(r'\W', ' ', text)  # Remove special characters
    text = re.sub(r'\s+', ' ', text)  # Remove extra whitespace
    text = text.lower()
```

```
    text = ' '.join([word for word in text.split() if word not in stop_words])
    return text

file_contents = [preprocess(doc) for doc in file_contents]

# Vectorization
vectorizer = TfidfVectorizer(max_features=1000)
X = vectorizer.fit_transform(file_contents).toarray()
```

```
[nltk_data] Downloading package stopwords to C:\Users\Ankit
[nltk_data]     Rajput\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

0.0.2 Implement K-Means Clustering

[5]:
```
inertia = []
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

plt.plot(range(1, 10), inertia, 'bx-')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()

# Fit the K-Means with optimal clusters
optimal_k = 5  # Replace with the number based on the Elbow Method
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans_labels = kmeans.fit_predict(X)

# Evaluate with Silhouette Score
silhouette_avg = silhouette_score(X, kmeans_labels)
print("K-Means Silhouette Score:", silhouette_avg)
```

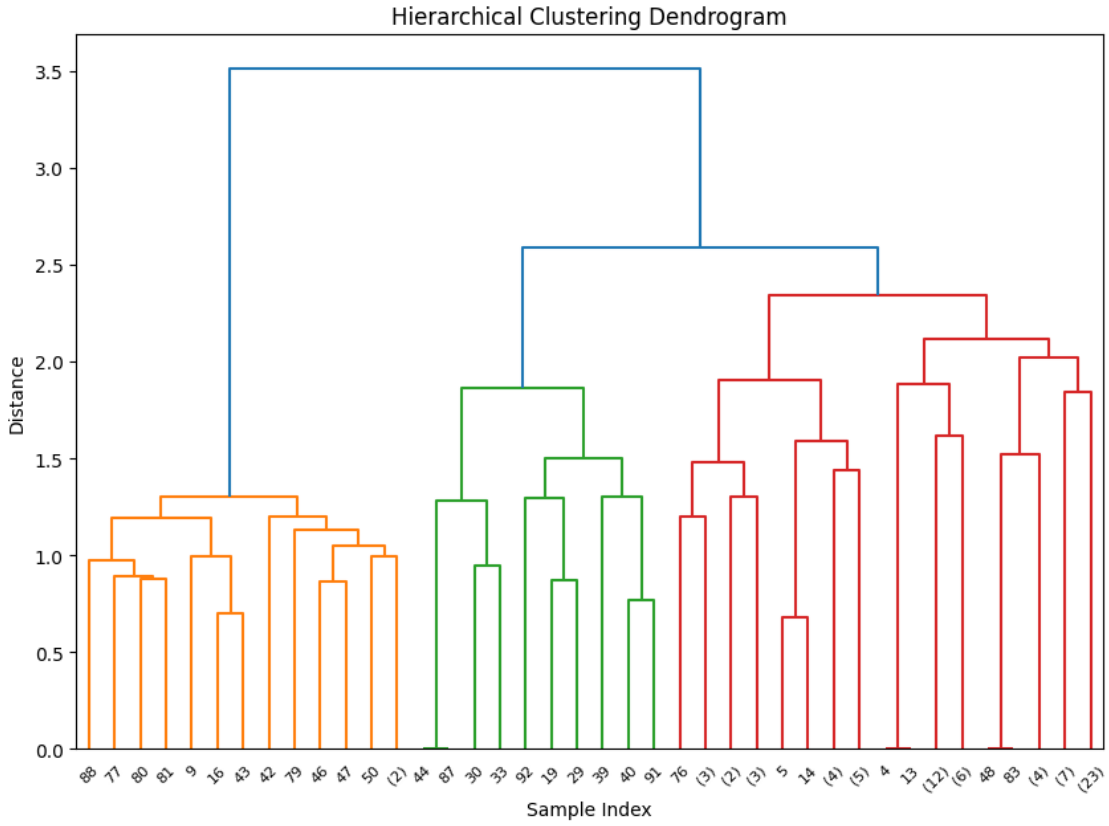



```
K-Means Silhouette Score: 0.043826224213418724
```

0.0.3 Implementing Hierarchical Agglomerative Clustering

[9]:
```
Z = linkage(X, 'ward')
plt.figure(figsize=(10, 7))
dendrogram(Z, truncate_mode='level', p=5)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

# agglomerative = AgglomerativeClustering(n_clusters=optimal_k,␣
 ↪affinity='euclidean', linkage='ward')
agglomerative = AgglomerativeClustering(n_clusters=optimal_k,␣
 ↪metric='euclidean', linkage='ward')

agglomerative_labels = agglomerative.fit_predict(X)

silhouette_avg_agglomerative = silhouette_score(X, agglomerative_labels)
```

```
print("Agglomerative Clustering Silhouette Score:",␣
  ↪silhouette_avg_agglomerative)
```




```
Agglomerative Clustering Silhouette Score: 0.0673568981189222
```

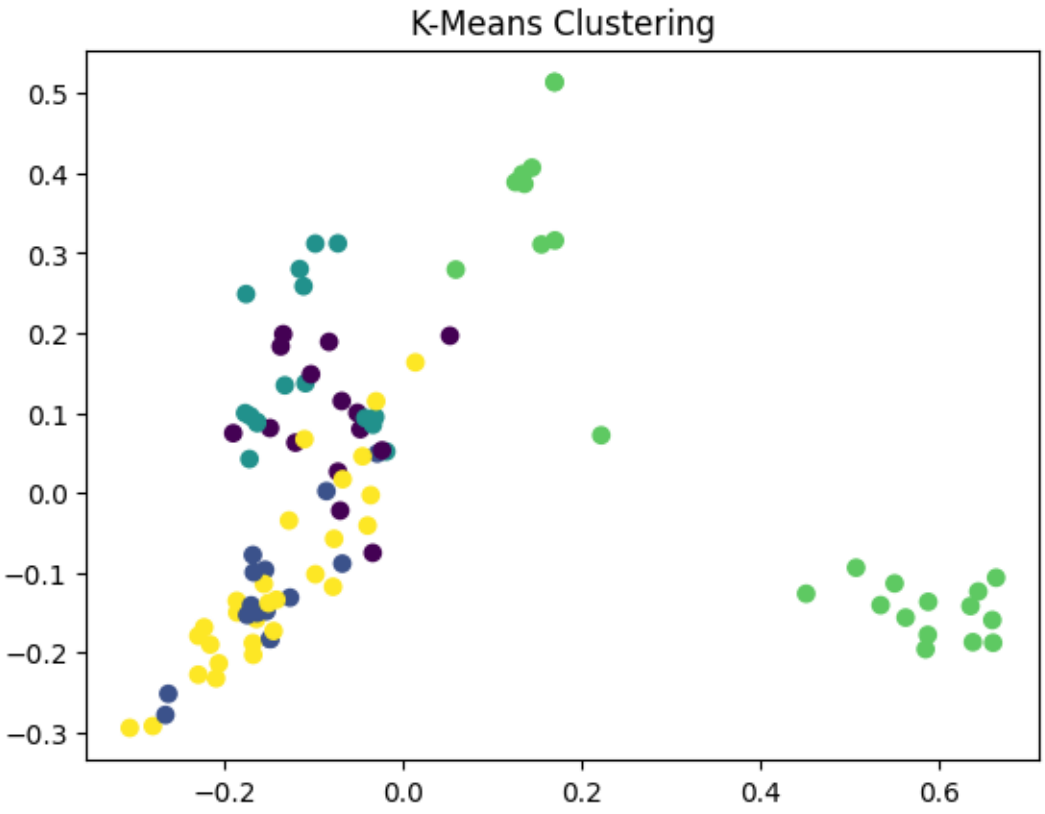
0.0.4 Implement Divisive Clustering

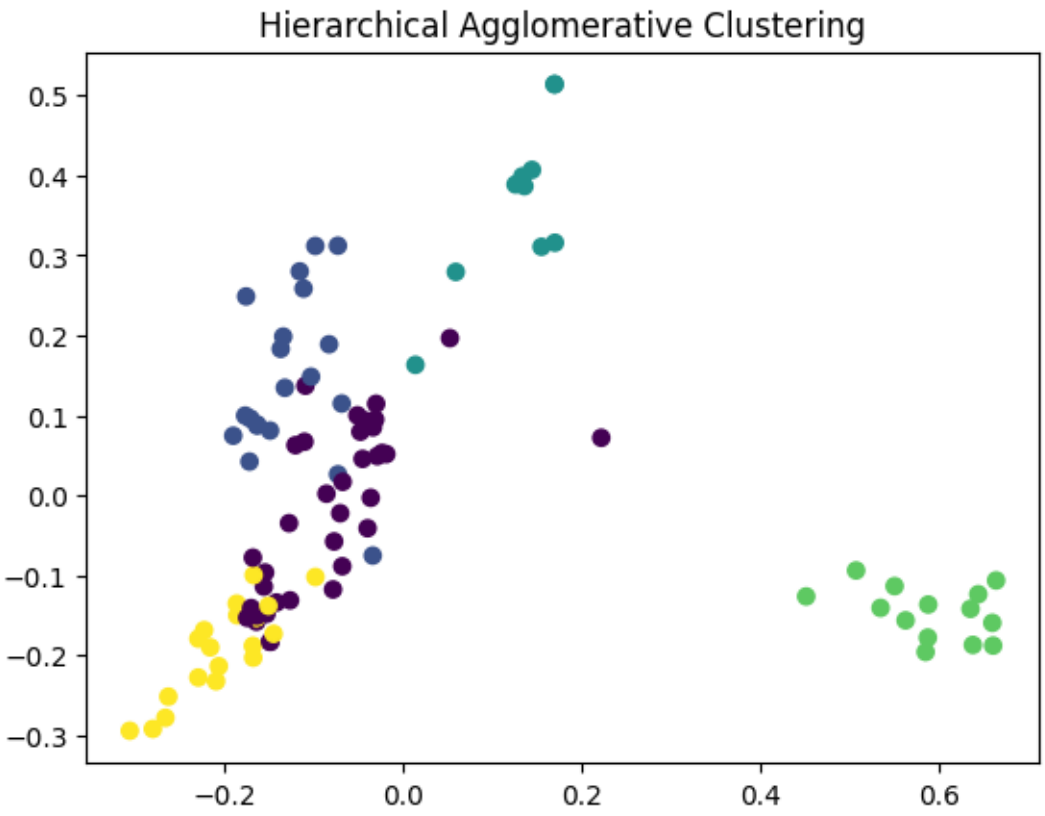
[7]:
```
from sklearn.cluster import BisectingKMeans

bisecting_kmeans = BisectingKMeans(n_clusters=optimal_k, random_state=42)
divisive_labels = bisecting_kmeans.fit_predict(X)

silhouette_avg_divisive = silhouette_score(X, divisive_labels)
print("Divisive Clustering (Bisecting K-Means) Silhouette Score:",␣
  ↪silhouette_avg_divisive)
```

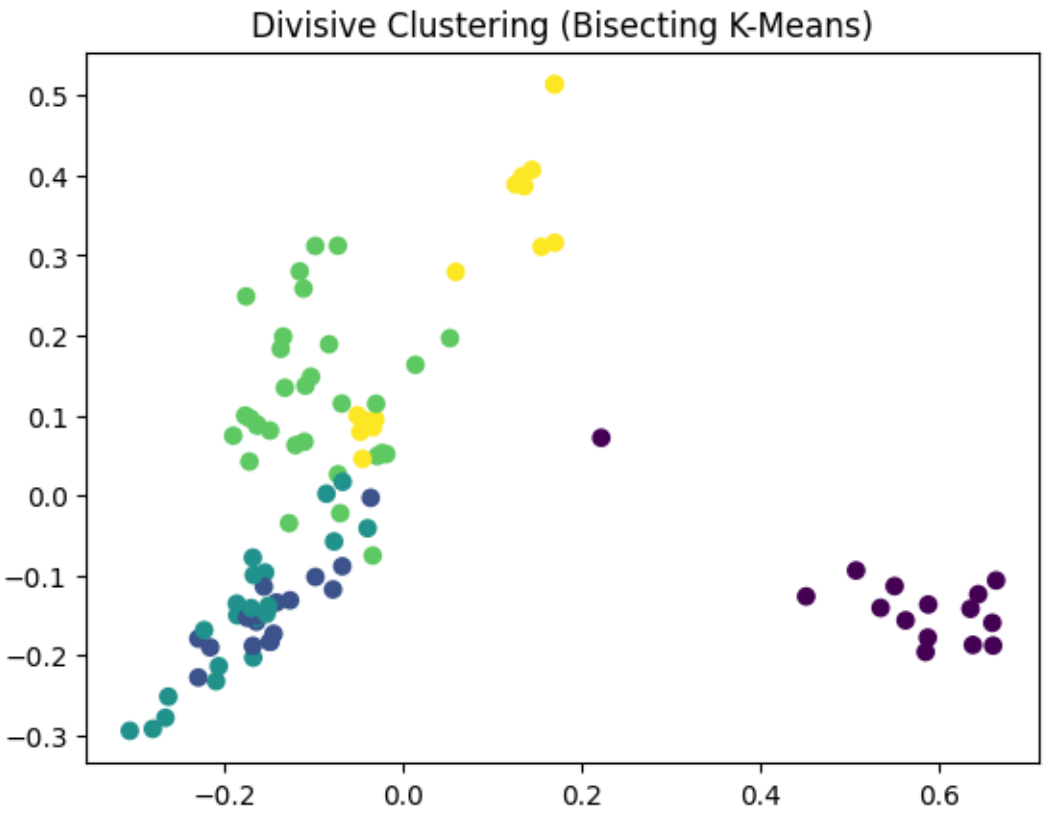
```
Divisive Clustering (Bisecting K-Means) Silhouette Score: 0.05864006806325925
```

0.0.5 Analyze Results

[10]:
```
from sklearn.decomposition import PCA

# Reduce dimensions for visualization
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=kmeans_labels, cmap='viridis')
plt.title('K-Means Clustering')
plt.show()

plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=agglomerative_labels,␣
 ↪cmap='viridis')
plt.title('Hierarchical Agglomerative Clustering')
plt.show()

plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=divisive_labels, cmap='viridis')
plt.title('Divisive Clustering (Bisecting K-Means)')
plt.show()
```




Hierarchical Agglomerative Clustering
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.2
0.0
0.2
0.4
0.6


Divisive Clustering (Bisecting K-Means)
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.2
0.0
0.2
0.4
0.6