Index

Sr. No.	*Name of the Practicals*	*Date*	*Sign.*
1	**Document Indexing and Retrieval** • Implement an inverted index construction algorithm. • Build a simple document retrieval system using the constructed index.	**07/11/25**	
2	**Retrieval Models** • Implement the Boolean retrieval model and process queries. • Implement the vector space model with TF-IDF weighting and cosine similarity.	**14/11/25**	
3	**Spelling Correction in IR Systems** • Develop a spelling correction module using edit distance algorithms. • Integrate the spelling correction module into an information retrieval system	**18/11/25**	
4	**Evaluation Metrics for IR Systems** • Calculate precision, recall, and F-measure for a given set of retrieval results. • Use an evaluation toolkit to measure average precision and other evaluation metrics	**15/12/25**	
5	**Text Categorization** • Implement a text classification algorithm (e.g., Naive Bayes or Support Vector Machines). • Train the classifier on a labelled dataset and evaluate its performance.	**19/12/25**	
6	**Clustering for Information Retrieval** • Implement a clustering algorithm (e.g., K-means or hierarchical clustering). • Apply the clustering algorithm to a set of documents and evaluate the clustering results.	**02/01/26**	
7	**Web Crawling and Indexing** • Develop a web crawler to fetch and index web pages. • Handle challenges such as robots.txt, dynamic content, and crawling delays.	**09/01/26**	

Sr. No.	*Name of the Practicals*	*Date*	*Sign.*
8	**Link Analysis and PageRank** • Implement the PageRank algorithm to rank web pages based on link analysis. • Apply the PageRank algorithm to a small web graph and analyze the results.	**16/01/26**	
9	**Learning to Rank** • Implement a learning to rank algorithm (e.g., RankSVM or RankBoost). • Train the ranking model using labelled data and evaluate its effectiveness.	**30/01/25**	
10	**Advanced Topics in Information Retrieval** • Implement a text summarization algorithm (e.g., extractive or abstractive). • Build a question-answering system using techniques such as information extraction	**06/02/25**	

Practical 1

Aim: Document Indexing and Retrieval

- **Implement an inverted index construction algorithm.**
- **Build a simple document retrieval system using the constructed index.**

Source Code:

```
index = {
    1: "This is the first document",
    2: "This document is the second document",
    3: "And this is the third one",
    4: "Is this the first document?",
    5: "The last document is here"
}
inverted_index ={}
for doc_id, text in index.items():
    for word in text.split():
        inverted_index.setdefault(word, set()).add(doc_id)
query = input("Enter your query: ")
result_docs=set()
for word in query.split():
    result_docs.update(inverted_index.get(word, []))
if result_docs:
    print("Relevant documnet found:")
    for doc_id in result_docs:
        print(f"Document {doc_id}: {index[doc_id]}")
else:
    print("No relevant documents found.")
```

Output:

```
Enter your query: this is the first document relevant document found
Relevant documnet found:
Document 1: This is the first document
Document 2: This document is the second document
Document 3: And this is the third one
Document 4: Is this the first document?
Document 5: The last document is here
```

Practical 2

Aim: **Retrieval Models**

- **Implement the Boolean retrieval model and process queries.**
- **Implement the vector space model with TF-IDF weighting and cosine similarity.**

Source Code:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def compute_similarity(doc1, doc2):
    tfidf_matrix = TfidfVectorizer().fit_transform([doc1, doc2])
    return cosine_similarity(tfidf_matrix[0], tfidf_matrix[1])[0][0]

doc1 = "This is the first document."
doc2 = "This document is the second document."

similarity = compute_similarity(doc1, doc2)
print("Similarity between the two documents:", similarity)
```

Output:

```
Similarity between the two documents: 0.6827531502984261
```

Practical 3

Aim: **Spelling Correction in IR Systems**

- **Develop a spelling correction module using edit distance algorithms.**
- **Integrate the spelling correction module into an information retrieval system.**

Source Code:

```
import numpy as np
class SpellingCorrection:
    def __init__(self):
        self.v = {"apple", "banana", "orange", "pear", "peach"}
    def ld(self, w1, w2):
        m, n = len(w1), len(w2)
        d = np.zeros((m + 1, n + 1), dtype=int)
        d[:, 0] = np.arange(m + 1)
        d[0, :] = np.arange(n + 1)
        for i in range(1, m + 1):
            for j in range(1, n + 1):
                cost = 0 if w1[i - 1] == w2[j - 1] else 1
                d[i][j] = min(d[i - 1][j] + 1,
                              d[i][j - 1] + 1,
                              d[i - 1][j - 1] + cost)
        return d[m][n]
    def correct(self, w):
        return min(self.v, key=lambda x: self.ld(w, x))
class IRS:
    def __init__(self, docs):
        self.d = docs
    def search(self, q):
        return [doc for doc in self.d if q in doc]
docs = ["An apple eats a day keep the doctor away", "banana contains
magnesium", "orange juice contains citric acid", "yellow pear",
"brown peach"]
q = "appl"
sc = SpellingCorrection()
irs = IRS(docs)
cq = sc.correct(q)
print("Corrected query:", cq)
sr = irs.search(cq)
print("Search results:", sr)
```

Output:

```
Corrected query: apple
Search results: ['An apple eats a day keep the doctor away']
```

Practical 4

Aim: Evaluation Metrics for IR Systems

- **Calculate precision, recall, and F-measure for a given set of retrieval results.**
- **Use an evaluation toolkit to measure average precision and other evaluation metrics.**

Source Code:

```
from sklearn.metrics import precision_score, recall_score, f1_score,
average_precision_score

relevant_documents = [1, 2, 3, 4, 5]
retrieved_documents = [1, 2, 3, 6, 7]

precision = len(set(relevant_documents).intersection(retrieved_documents)) /
len(retrieved_documents)
recall = len(set(relevant_documents).intersection(retrieved_documents)) /
len(relevant_documents)
f_measure = (
    2 * (precision * recall) / (precision + recall)
    if precision + recall > 0 else 0
)

print("Manual Calculation:")
print("Precision:", precision)
print("Recall:", recall)
print("F-measure:", f_measure)

y_true = [1 if doc in relevant_documents else 0 for doc in retrieved_documents]
y_pred = [1 if doc in retrieved_documents else 0 for doc in
relevant_documents]

sklearn_precision = precision_score(y_true, y_pred)
sklearn_recall = recall_score(y_true, y_pred)
sklearn_f1 = f1_score(y_true, y_pred)

print("\nEvaluation Toolkit (sklearn):")
print("Precision (sklearn):", sklearn_precision)
print("Recall (sklearn):", sklearn_recall)
print("F1 Score (sklearn):", sklearn_f1)

y_scores = [1 if doc in relevant_documents else 0 for doc in
retrieved_documents]
avg_precision = average_precision_score(y_true, y_scores)
```

```
print("Average Precision:", avg_precision)
```

Output:

```
Manual Calculation:
Precision: 0.6
Recall: 0.6
F-measure: 0.6

Evaluation Toolkit (sklearn):
Precision (sklearn): 1.0
Recall (sklearn): 1.0
F1 Score (sklearn): 1.0
Average Precision: 1.0
```

Practical 5

Aim: Text Categorization

- **Implement a text classification algorithm (e.g., Naive Bayes or Support Vector Machines).**
- **Train the classifier on a labelled dataset and evaluate its performance.**

Source Code:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split

X = [
    "I love this movie",
    "This movie is terrible",
    "Great movie, highly recommended",
    "Waste of time, not worth watching",
    "Amazing performance by the actors"
]
y = ["Positive", "Negative", "Positive", "Negative", "Positive"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=42, stratify=y
)
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

svm_model = SVC(kernel='linear')
svm_model.fit(X_train_tfidf, y_train)

y_pred = svm_model.predict(X_test_tfidf)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred, zero_division=0))
```

Output:

```
Accuracy: 0.5
              precision    recall  f1-score   support

    Negative       0.00      0.00      0.00         1
    Positive       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

Practical 6

Aim: Clustering for Information Retrieval

- **Implement a clustering algorithm (e.g., K-means or hierarchical clustering).**
- **Apply the clustering algorithm to a set of documents and evaluate the clustering results.**

Source Code:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

documents = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?",
    "This is the last document."
    ]

last_document = documents[-1]

tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(documents)

k = 2

if tfidf_matrix.shape[0] >= k:
    kmeans = KMeans(n_clusters=k, random_state=42)
    clusters = kmeans.fit_predict(tfidf_matrix)
    silhouette_score_value = silhouette_score(tfidf_matrix, clusters)
    print("Silhouette Score:", silhouette_score_value)
else:
    print("Error: Number of clusters exceeds the number of documents.")
```

Output:

```
Silhouette Score: 0.07253699431826806
```

Practical 7

Aim: Web Crawling and Indexing

- **Develop a web crawler to fetch and index web pages.**
- **Handle challenges such as robots.txt, dynamic content, and crawling delays.**

Source Code:

```
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin, urlparse
import time
import re
import urllib.robotparser as robotparser

class WebCrawler:
    def __init__(self, seed_url, max_pages=10, delay=2):
        self.seed_url = seed_url
        self.max_pages = max_pages
        self.delay = delay
        self.visited_urls = set()
        self.index = {}
        self.robot_parser = robotparser.RobotFileParser()
        robots_url = urljoin(seed_url, "/robots.txt")
        self.robot_parser.set_url(robots_url)
        self.robot_parser.read()

    def is_allowed(self, url):
        """Check robots.txt permission"""
        return self.robot_parser.can_fetch("*", url)

    def fetch_page(self, url):
        """Fetch webpage content"""
        try:
            response = requests.get(url, timeout=10, headers={
                "User-Agent": "SimpleWebCrawler/1.0"
            })
            if response.status_code == 200:
                return response.text
        except requests.exceptions.RequestException:
            return None
        return None
```

```
def extract_text(self, html):
    """Extract visible text from HTML"""
    soup = BeautifulSoup(html, "lxml")
    for script in soup(["script", "style"]):
        script.decompose()
    text = soup.get_text(separator=" ")
    text = re.sub(r"\s+", " ", text)
    return text.strip()

def extract_links(self, html, base_url):
    """Extract and normalize hyperlinks"""
    soup = BeautifulSoup(html, "lxml")
    links = set()
    for tag in soup.find_all("a", href=True):
        full_url = urljoin(base_url, tag["href"])
        parsed = urlparse(full_url)
        if parsed.scheme in ["http", "https"]:
            links.add(full_url)
    return links

def crawl(self):
    """Start crawling process"""
    urls_to_crawl = [self.seed_url]
    while urls_to_crawl and len(self.visited_urls) < self.max_pages:
        current_url = urls_to_crawl.pop(0)
        if current_url in self.visited_urls:
            continue
        if not self.is_allowed(current_url):
            print(f"Blocked by robots.txt: {current_url}")
            continue
        print(f"Crawling: {current_url}")
        html = self.fetch_page(current_url)
        if html:
            text = self.extract_text(html)
            self.index[current_url] = text
            self.visited_urls.add(current_url)
            links = self.extract_links(html, current_url)
            for link in links:
                if link not in self.visited_urls:
                    urls_to_crawl.append(link)
        time.sleep(self.delay)
    print("\nCrawling completed!")
    print(f"Total pages indexed: {len(self.index)}")
```

```
    def display_index(self):
        """Display indexed pages"""
        for url, content in self.index.items():
            print("\nURL:", url)
            print("Indexed Content (first 300 chars):")
            print(content[:300])

if __name__ == "__main__":
    seed_url = "https://example.com"
    crawler = WebCrawler(seed_url, max_pages=5, delay=2)
    crawler.crawl()
    crawler.display_index()
```

Output:

```
Crawling: https://example.com
Crawling: https://iana.org/domains/example
Crawling: https://www.icann.org/
Crawling: https://iana.org/domains/root
Crawling: https://iana.org/news
Crawling: https://iana.org/go/rfc6761

Crawling completed!
Total pages indexed: 5

URL: https://example.com
Indexed Content (first 300 chars):
Example Domain Example Domain This domain is for use in documentation examples without
needing permission. Avoid use in operations. Learn more

URL: https://iana.org/domains/example
Indexed Content (first 300 chars):
Example Domains Domains Protocols Numbers About Example Domains As described in RFC 26
06 and RFC 6761 , a number of domains such as example.com and example.org are maintain
ed for documentation purposes. These domains may be used as illustrative examples in d
ocuments without prior coordination with u

URL: https://iana.org/domains/root
Indexed Content (first 300 chars):
Root Zone Management Domains Protocols Numbers About Root Zone Management We are respo
nsible for management of the DNS root zone. This role means assigning the operators of
top-level domains, such as .uk and .com , and maintaining their technical and administ
rative details. Browse The Root Zone Data

URL: https://iana.org/news
Indexed Content (first 300 chars):
News Domains Protocols Numbers About News Atom feed All Update Blog Oversight Publicat
ion Consultation Blog 28 April 2025 PTI Board Adopts 2025-30 Strategic Plan and FY26 O
perating Plan and Budget PTI's five year strategic plan set out objectives for the nex
t five years for the IANA functions. Consu

URL: https://iana.org/go/rfc6761
Indexed Content (first 300 chars):
RFC 6761: Special-Use Domain Names [ RFC Home ] [ TEXT | PDF | HTML ] [ Tracker ] [ IP
R ] [ Errata ] [ Info page ] PROPOSED STANDARD Errata Exist Internet Engineering Task
Force (IETF) S. Cheshire Request for Comments: 6761 M. Krochmal Updates: 1918 , 2606 A
pple Inc. Category: Standards Track Februa
```

Practical 8

Aim: **Link Analysis and PageRank**

- **Implement the PageRank algorithm to rank web pages based on link analysis.**
- **Apply the PageRank algorithm to a small web graph and analyze the results.**

Source Code:

```
import numpy as np

def pagerank(adj_matrix, damping_factor=0.85, max_iterations=100,
epsilon=1.0e-6):
    n = len(adj_matrix)
    teleport_prob = (1 - damping_factor) / n
    scores = np.ones(n) / n

    for _ in range(max_iterations):
        prev_scores = np.copy(scores)
        scores = teleport_prob + damping_factor * np.dot(adj_matrix.T, scores)
        if np.linalg.norm(scores - prev_scores) < epsilon:
            break

    return scores

adj_matrix = np.array([
    [0, 1, 1, 0],
    [1, 0, 1, 1],
    [1, 1, 0, 1],
    [0, 1, 1, 0]
])

pagerank_scores = pagerank(adj_matrix)
print("PageRank Scores:", pagerank_scores)
```

Output:

```
PageRank Scores: [1.50889864e+33 1.93256177e+33 1.93256177e+33 1.50889864e+33]
```

Practical 9

Aim: **Learning to Rank**

- **Implement a learning to rank algorithm (e.g., RankSVM or RankBoost).**
- **Train the ranking model using labelled data and evaluate its effectiveness.**

Source Code:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error

data = load_iris()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

rank_model = GradientBoostingRegressor(
n_estimators=100, learning_rate=0.1, max_depth=3
)

rank_model.fit(X_train, y_train)

y_pred = rank_model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

Output:

```
Mean Squared Error: 0.00465466900622937
```

Practical 10

Aim: Advanced Topics in Information Retrieval

- **Implement a text summarization algorithm (e.g., extractive or abstractive).**
- **Build a question-answering system using techniques such as information extraction.**

Source Code:

```
import spacy
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from heapq import nlargest
nlp = spacy.load("en_core_web_sm")
def summarize_text(text, num_sentences=3):
    """
    Summarizes the input text by identifying the most relevant sentences.
    Args:
        text (str): The text to be summarized.
        num_sentences (int): The number of sentences to include in the summary.
    Returns:
        str: The summary of the text.
    """
    doc = nlp(text)
    sentences = [sent.text for sent in doc.sents]
    word_frequency = {}
    for word in doc:
        if not word.is_stop and word.is_alpha:
            key = word.text.lower()
            word_frequency[key] = word_frequency.get(key, 0) + 1
    sentence_scores = {}
    for sentence in sentences:
        for word in sentence.split():
            word_lower = word.lower()
            if word_lower in word_frequency:
                sentence_scores[sentence] = sentence_scores.get(sentence, 0) +
word_frequency[word_lower]
    summary_sentences = nlargest(num_sentences, sentence_scores,
key=sentence_scores.get)
    summary = " ".join(summary_sentences)
    return summary
def answer_question(question, text):
    """
    Answers a question by finding the most similar sentence in the text.
    Args:
        question (str): The question to be answered.
```

```
        text (str): The text to search for the answer.
    Returns:
        str: The most relevant sentence from the text.
    """
    sentences = text.split(".")
    vectorizer = TfidfVectorizer(stop_words="english")
    vectors = vectorizer.fit_transform(sentences + [question])
    similarity = cosine_similarity(vectors[-1], vectors[:-1])[0]
    best_sentence_index = similarity.argmax()
    return sentences[best_sentence_index].strip()
text = """
In Speed Rush, the latest high-octane racing game, players are immersed
in a world of adrenaline-pumping tracks and custom cars. Experience
breathtaking
graphics, realistic physics, and intense rivalries as you climb the leaderboards.
Master the art of drifting, boost management, and strategic shortcuts to outpace
your competition.
With an array of multiplayer modes and seasonal challenges,
Speed Rush keeps the excitement alive. Do you have what it takes to become
the
ultimate racing champion?
"""
summary = summarize_text(text)
print("Game Text Summary:")
print(summary)
question = "What is Speed Rush about?"
answer = answer_question(question, text)
print("\nQuestion:", question)
print("Answer:", answer)
```

Output:

```
Game Text Summary:

In Speed Rush, the latest high-octane racing game, players are immersed
in a world of adrenaline-pumping tracks and custom cars. With an array of multiplayer
modes and seasonal challenges,
Speed Rush keeps the excitement alive. Experience breathtaking
graphics, realistic physics, and intense rivalries as you climb the leaderboards.


Question: What is Speed Rush about?
Answer: With an array of multiplayer modes and seasonal challenges,
Speed Rush keeps the excitement alive
```