

# Python Solutions for Practical Questions

## Q1A, Q6A, Q7A, Q11A: Naive Bayes Classifier for Positive/Negative Sentiment

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Load subset of 20_newsgroups (e.g., sports as positive, atheism as negative)
categories = ['rec.sport.baseball', 'alt.atheism']
newsgroups = fetch_20newsgroups(subset='all', categories=categories, remove=('headers',

# Convert text to TF-IDF features
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(newsgroups.data)
y = newsgroups.target # Labels: 0 (atheism), 1 (sports)

# Split data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# Train Naive Bayes classifier
nb = MultinomialNB()
nb.fit(X_train, y_train)

# Predict and evaluate
y_pred = nb.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

## Q1B, Q6B, Q7B, Q11B: SVM Classifier for Multi-class Categorization

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Load 20_newsgroups for specified categories
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']
newsgroups = fetch_20newsgroups(subset='all', categories=categories, remove=('headers',

# Convert text to TF-IDF features
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(newsgroups.data)
y = newsgroups.target

# Split data
```

```

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

# Train SVM classifier
svm = LinearSVC()
svm.fit(X_train, y_train)

# Predict and evaluate
y_pred = svm.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))

```

## Q2A, Q4A: PageRank Algorithm for Web Graph

```

import numpy as np

# Define adjacency matrix (rows = source, cols = target)
# Pages: A, B, C, D, E
adj_matrix = np.array([
    [0, 1, 1, 1, 0], # A -> B, C, D
    [0, 0, 1, 0, 1], # B -> C, E
    [1, 0, 0, 1, 0], # C -> A, D
    [0, 0, 0, 0, 0], # D -> none
    [0, 0, 0, 0, 0]  # E -> none
])

# Normalize to get transition probabilities
n = adj_matrix.shape[0]
transition_matrix = np.zeros_like(adj_matrix, dtype=float)
for i in range(n):
    out_links = adj_matrix[i].sum()
    if out_links > 0:
        transition_matrix[i] = adj_matrix[i] / out_links

# PageRank parameters
d = 0.85 # Damping factor
max_iter = 100
epsilon = 1e-6
pr = np.ones(n) / n # Initial PageRank

# Power iteration
for _ in range(max_iter):
    pr_new = (1 - d) / n + d * transition_matrix.T @ pr
    if np.linalg.norm(pr_new - pr) < epsilon:
        break
    pr = pr_new

# Print PageRank scores
pages = ['A', 'B', 'C', 'D', 'E']
for i, score in enumerate(pr):
    print(f"Page {pages[i]}: {score:.4f}")

```

## Q2B: Text Summarization (Extractive)

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import numpy as np

# Sample text
text = """Machine learning is a field of artificial intelligence. It enables computers to learn from data.
Deep learning is a subset of machine learning. It uses neural networks for complex tasks.

# Preprocess
stop_words = set(stopwords.words('english'))
ps = PorterStemmer()

def score_sentence(sent, word_freq):
    words = word_tokenize(sent.lower())
    return sum(word_freq.get(ps.stem(w), 0) for w in words if w not in stop_words)

# Tokenize and compute word frequency
sents = sent_tokenize(text)
words = word_tokenize(text.lower())
word_freq = {ps.stem(w): words.count(w) for w in words if w not in stop_words}

# Score sentences
scores = [score_sentence(sent, word_freq) for sent in sents]
top_n = 2 # Select top 2 sentences
top_indices = np.argsort(scores)[-top_n:]

# Generate summary
summary = [sents[i] for i in sorted(top_indices)]
print("Summary:", " ".join(summary))
```

## Q3A, Q5A, Q5B: K-means Clustering for Documents

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.datasets import fetch_20newsgroups

# Load documents (use 20_newsgroups for Q5B, or provided documents for Q3A)
documents = [
    "Machine learning is the study of computer algorithms that improve through experience.",
    "Deep learning is a subset of machine learning.",
    "Natural language processing is a field of artificial intelligence.",
    # ... (add other documents from Q3A)
]

# For Q5B, use 20_newsgroups
newsgroups = fetch_20newsgroups(subset='all', categories=['sci.med', 'comp.graphics'], shuffle=True)
documents = newsgroups.data[:100] # Limit for simplicity

# Convert to TF-IDF
```

```

vectorizer = TfidfVectorizer(max_features=1000, stop_words='english')
X = vectorizer.fit_transform(documents)

# Apply K-means
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X)

# Print cluster assignments
labels = kmeans.labels_
for i, doc in enumerate(documents[:5]): # Show first 5 for brevity
    print(f"Doc {i}: Cluster {labels[i]}")

```

### Q3B: Evaluate Binary Classification Metrics

```

from sklearn.metrics import precision_score, recall_score, f1_score, average_precision_score
import numpy as np

# Provided data
y_true = [0, 1, 1, 0, 1]
y_scores = [0.1, 0.8, 0.6, 0.3, 0.9]
y_pred = [1 if score >= 0.5 else 0 for score in y_scores] # Threshold at 0.5

# Calculate metrics
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
avg_precision = average_precision_score(y_true, y_scores)

print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
print(f"Average Precision: {avg_precision:.4f}")

```

### Q4B: Calculate Precision, Recall, and F-measure

```

# Provided values
TP = 20
FP = 10
FN = 30

# Calculate metrics
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f_score = 2 * (precision * recall) / (precision + recall)

print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F-measure: {f_score:.4f}")

```

## Q8A, Q10A: Vector Space Model with TF-IDF Weighting

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Corpus and query
corpus = [
    "Document about python programming language and data analysis.",
    "Document discussing machine learning algorithms and programming techniques.",
    "Overview of natural language processing and its applications."
]
query = ["python programming"]

# Compute TF-IDF
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus + query) # Include query in vectorization
tfidf_matrix = X[:-1] # Documents
query_vector = X[-1] # Query

# Print TF-IDF matrix (for documents)
print("TF-IDF Matrix for Documents:")
print(tfidf_matrix.toarray())
```

## Q8B, Q10B, Q12A: Cosine Similarity

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

# Q8B, Q10A corpus
corpus = [
    "Document about python programming language and data analysis.",
    "Document discussing machine learning algorithms and programming techniques.",
    "Overview of natural language processing and its applications."
]
query = ["python programming"]

# Q10B, Q12A case
# corpus = ["shipment of gold damaged in a gold fire"]
# query = ["gold silver truck"]

# Compute TF-IDF
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus + query)
doc_vectors = X[:-1]
query_vector = X[-1]

# Calculate cosine similarity
similarities = cosine_similarity(query_vector, doc_vectors)[0]
for i, sim in enumerate(similarities):
    print(f"Similarity with Document {i+1}: {sim:.4f}")
```

## Q9A: Question-Answering System

```
from nltk.tokenize import word_tokenize, sent_tokenize
import re

# Corpus
corpus = [
    "The cat is on the mat.",
    "The dog is in the yard.",
    "A bird is flying in the sky.",
    "The sun is shining brightly."
]
query = "Where is the cat?"

# Simple QA: Find sentence with "cat" and extract location
for sent in corpus:
    if "cat" in sent.lower():
        # Extract location using regex or simple parsing
        location = re.search(r"(on|in|at) ([\w\s]+\.)", sent.lower())
        if location:
            print(f"Answer: The cat is {location.group(0)}")
            break
    else:
        print("Answer: Not found")
```

## Q9B: Inverted Index and Document Retrieval

```
from collections import defaultdict
from nltk.tokenize import word_tokenize

# Documents
docs = [
    "best of luck tycs students for your practical examination.",
    "tycs students please carry your journal at the time of practical examination."
]
query = "tycs journal"

# Build inverted index
inverted_index = defaultdict(list)
for doc_id, doc in enumerate(docs):
    words = word_tokenize(doc.lower())
    for word in set(words): # Avoid duplicates
        inverted_index[word].append(doc_id)

# Query processing
query_words = word_tokenize(query.lower())
results = set.intersection(*[set(inverted_index[word]) for word in query_words if word])

# Print results
print("Documents containing all query terms:", list(results))
```

## Q12B: Question-Answering System (General)

```
from nltk.tokenize import word_tokenize, sent_tokenize
import re

# Sample text
text = """The cat is on the mat. The dog is in the yard."""

query = "Where is the dog?"

# Tokenize sentences
sents = sent_tokenize(text)

# Find relevant sentence and extract location
for sent in sents:
    if "dog" in sent.lower():
        location = re.search(r"(on|in|at) ([\w\s]+\.)", sent.lower())
        if location:
            print(f"Answer: The dog is {location.group(0)}")
            break
else:
    print("Answer: Not found")
```

## Q13A: Spelling Correction with Edit Distance

```
def edit_distance(s1, s2):
    m, n = len(s1), len(s2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    # Initialize first row and column
    for i in range(m + 1):
        dp[i][0] = i
    for j in range(n + 1):
        dp[0][j] = j

    # Fill dp table
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if s1[i-1] == s2[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1]) + 1

    return dp[m][n]

# Test
s1, s2 = "write", "right"
print(f"Edit distance between '{s1}' and '{s2}': {edit_distance(s1, s2)}")
```

## Q13B: Clustering Algorithm (K-means) and Evaluation

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

# Documents
documents = [
    "Machine learning is the study of computer algorithms that improve through experience.",
    "Deep learning is a subset of machine learning.",
    "Natural language processing is a field of artificial intelligence.",
    # ... (add other documents)
]

# Convert to TF-IDF
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)

# Apply K-means
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X)
labels = kmeans.labels_

# Evaluate with silhouette score
sil_score = silhouette_score(X, labels)
print(f"Silhouette Score: {sil_score:.4f}")

# Print cluster assignments
for i, doc in enumerate(documents):
    print(f"Doc {i}: Cluster {labels[i]}")
```