Simplified Lab Programs for Exam

This document contains simplified versions of all lab programs that use only built-in Python libraries. These programs are designed to be easy to understand and implement during exams without internet access.

Programs Included:

1. EXP4 - Association Rules Mining (Apriori Algorithm)
2. EXP5 - Sentiment Analysis using Naive Bayes
3. EXP6 - Web Scraping and Topic Modeling
4. EXP7 - HTML Sentiment Analysis
5. EXP8 - LDA Topic Modeling

EXP4 - Association Rules Mining (Apriori Algorithm)

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv("Market_Basket_Optimisation - Market_Basket_Optimisation.csv")

# Convert to transactions
transactions = []
for i in range(len(df)):
    transactions.append([str(df.values[i,j]) for j in range(df.shape[1]) if str(df.values[i,j]) != 'nan'

# Encode transactions
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Find frequent itemsets
frequent_itemsets = apriori(df_encoded, min_support=0.01, use_colnames=True)
print("Frequent Itemsets:", frequent_itemsets.shape[0])

# Generate rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.1)
rules = rules[rules['antecedents'].apply(lambda x: len(x) >= 1) & rules['consequents'].apply(lambda x: l
print("Association Rules:", rules.shape[0])

# Show top rules
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head(10))

# Top items
all_items = [item for sublist in transactions for item in sublist]
top_items = pd.Series(all_items).value_counts().head(10)
print("\nTop 10 Items:")
print(top_items)
```

EXP5 - Sentiment Analysis using Naive Bayes

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load data
df = pd.read_csv("customer_review - customer_review.csv")

# Auto-detect columns
text_col = None
label_col = None
for col in df.columns:
    if any(x in col.lower() for x in ['review', 'text', 'comment']):
        text_col = col
    if any(x in col.lower() for x in ['sentiment', 'label', 'rating']):
        label_col = col

# Prepare data
X = df[text_col].astype(str)
y = df[label_col].apply(lambda x: "Positive" if x > 2 else ("Negative" if x <= 2 else "Neutral"))

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

# Vectorize
vectorizer = TfidfVectorizer(stop_words='english', max_features=10000)
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Train model
clf = MultinomialNB()
clf.fit(X_train_vec, y_train)

# Predict
y_pred = clf.predict(X_test_vec)

# Results
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

EXP6 - Web Scraping and Topic Modeling

```
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF
import pandas as pd

# Files to process
files = ["Artificial intelligence/Artificialintelligence.html", "Startups_TechCrunch/Startups_TechCrunch

def extract_text(filepath):
    with open(filepath, 'r', encoding='utf-8') as f:
        soup = BeautifulSoup(f.read(), 'html.parser')
    for tag in soup(["script", "style", "noscript"]):
        tag.decompose()
    return soup.get_text()

# Extract text
docs = [extract_text(f) for f in files if len(extract_text(f)) > 200]

# TF-IDF
vectorizer = TfidfVectorizer(stop_words='english', ngram_range=(1,2), max_df=0.9, min_df=1)
X = vectorizer.fit_transform(docs)
terms = vectorizer.get_feature_names_out()
scores = X.mean(axis=0).A1

# Top terms
top_idx = scores.argsort()[::-1][:25]
print("Top TF-IDF terms:")
for i, idx in enumerate(top_idx):
    print(f"{terms[idx]:30s} {scores[idx]:.4f}")

# Topic modeling
k = min(6, len(docs))
nmf = NMF(n_components=k, random_state=42)
W = nmf.fit_transform(X)
H = nmf.components_

print("\nTopics:")
for i, topic in enumerate(H):
    top_words = topic.argsort()[::-1][:10]
    print(f"Topic {i}: {', '.join(terms[top_words])}")
```

EXP7 - HTML Sentiment Analysis

```
from bs4 import BeautifulSoup
import re
from collections import Counter

# Sentiment lexicon
positive_words = {'good', 'great', 'excellent', 'amazing', 'love', 'like', 'nice', 'awesome', 'helpful',
negative_words = {'bad', 'terrible', 'awful', 'hate', 'slow', 'buggy', 'confusing', 'broken', 'issue', '
negation_words = {'not', 'no', 'never', 'none', 'hardly', 'barely', 'scarcely'}

def extract_comments(html_file):
    with open(html_file, 'r', encoding='utf-8') as f:
        soup = BeautifulSoup(f.read(), 'html.parser')

    # Remove scripts and styles
    for tag in soup(["script", "style", "noscript", "iframe", "svg"]):
        tag.decompose()

    # Find comment elements
    comments = []
    for elem in soup.find_all(['div', 'p', 'span', 'article']):
        if any(word in elem.get('class', []) for word in ['comment', 'reply', 'review']):
            text = elem.get_text().strip()
            if len(text) > 12:
                comments.append(text)

    return comments

def analyze_sentiment(text):
    words = re.findall(r'\b\w+\b', text.lower())
    score = 0
    for i, word in enumerate(words):
        if word in positive_words:
            score += 1
        elif word in negative_words:
            score -= 1

        # Check for negation
        if i > 0 and words[i-1] in negation_words:
            score *= -1

    return "positive" if score > 0 else ("negative" if score < 0 else "neutral")

# Process file
comments = extract_comments("r_news/r_news.html")
if not comments:
    print("No comments found")
else:
    sentiments = [analyze_sentiment(comment) for comment in comments]
    counts = Counter(sentiments)

    print(f"Overall tone: {max(counts, key=counts.get)}")
    print(f"Counts: {dict(counts)}")

    # Show examples
    print("\nSample comments:")
    for i, comment in enumerate(comments[:5]):
        print(f"{i+1}. [{sentiments[i]}] {comment[:100]}...")
```

EXP8 - LDA Topic Modeling

```
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import re

# Files to process
files = ["The Indian Express/TheIndianExpress.html", "The Economic Times/TheEconomicTimes.html", "TIE/TI

def extract_text(filepath):
    with open(filepath, 'r', encoding='utf-8') as f:
        soup = BeautifulSoup(f.read(), 'html.parser')

    # Remove scripts and styles
    for tag in soup(["script", "style"]):
        tag.decompose()

    # Extract text from paragraphs
    text = " ".join(p.get_text() for p in soup.find_all('p'))
    text = re.sub(r'https?://\S+|www\.\S+', ' ', text)
    text = re.sub(r'\s+', ' ', text).strip()

    return text

# Extract texts
texts = [extract_text(f) for f in files if len(extract_text(f)) > 50]

# Create bag-of-words
vectorizer = CountVectorizer(stop_words='english', min_df=2, max_df=0.95)
X = vectorizer.fit_transform(texts)
vocab = vectorizer.get_feature_names_out()

# Find common terms
freq = X.sum(axis=0).A1
top_idx = freq.argsort()[::-1][:15]
print("Top Common Terms:")
for i, idx in enumerate(top_idx):
    print(f"{vocab[idx]:20s} {int(freq[idx])}")

# LDA Topic Modeling
k = min(3, len(texts))
lda = LatentDirichletAllocation(n_components=k, random_state=0)
lda.fit(X)

print("\nHidden Topics (LDA):")
for i, topic in enumerate(lda.components_):
    top_words = topic.argsort()[::-1][:10]
    print(f"Topic {i+1}: {' | '.join(vocab[top_words])}")
```

Documentation

Simplified Lab Programs Documentation

Overview

This document contains simplified versions of all lab programs that use only built-in Python libraries. These programs are designed to be easy to understand and implement during exams without internet access.

Programs Included

1. EXP4 - Association Rules Mining (Apriori Algorithm)

File: `EXP4_Simplified.py`
Purpose: Find frequent itemsets and association rules from market basket data
Data: `Market_Basket_Optimisation - Market_Basket_Optimisation.csv`
Key Features:
- Implements Apriori algorithm from scratch
- Finds frequent itemsets of different sizes
- Generates association rules with support, confidence, and lift
- Shows top purchased items
- Uses only `csv`, `collections`, `itertools`, and `math` libraries
How to Run:
```bash
python EXP4_Simplified.py
```
Expected Output:
- Top 10 most purchased items
- Total frequent itemsets found
- Association rules with metrics
- Rules sorted by confidence

2. EXP5 - Sentiment Analysis using Naive Bayes

File: `EXP5_Simplified.py`
Purpose: Analyze sentiment of customer reviews using Naive Bayes classifier
Data: `customer_review - customer_review.csv`
Key Features:
- Custom Naive Bayes implementation
- Text preprocessing and tokenization
- Train-test split functionality
- Confusion matrix display
- Interactive testing mode
- Uses only built-in libraries
How to Run:
```bash
python EXP5_Simplified.py
```
Expected Output:
- Class distribution
- Training and test sample counts
- Accuracy score

- Confusion matrix
- Sample predictions
- Interactive testing interface

3. EXP6 - Web Scraping and Topic Modeling

File: `EXP6_Simplified.py`
Purpose: Extract text from HTML files and perform topic modeling
Data: HTML files in `Artificial intelligence/` and `Startups_TechCrunch/` folders
Key Features:
- Custom HTML parser
- Text extraction and cleaning
- TF-IDF calculation
- Simple topic modeling using word co-occurrence
- Stopword removal
- Uses only built-in libraries
How to Run:
```bash
python EXP6_Simplified.py
```
Expected Output:
- Processing status for each HTML file
- Top TF-IDF terms (trends)
- Simple topic modeling results
- Document statistics
- Most common words overall

4. EXP7 - HTML Sentiment Analysis

File: `EXP7_Simplified.py`
Purpose: Extract comments from HTML files and analyze their sentiment
Data: `r_news/r_news.html`
Key Features:
- Comment extraction from HTML
- Lexicon-based sentiment analysis
- Negation handling
- Intensifier detection
- Sentiment distribution analysis
- Interactive testing
- Uses only built-in libraries
How to Run:
```bash
python EXP7_Simplified.py
```
Expected Output:
- Number of comments found
- Overall sentiment tone
- Sentiment distribution
- Sample positive and negative comments
- Interactive testing interface

5. EXP8 - LDA Topic Modeling

File: `EXP8_Simplified.py`
Purpose: Perform topic modeling on news articles using simplified LDA
Data: HTML files in `The Indian Express/`, `The Economic Times/`, and `TIE/` folders
Key Features:

- Custom LDA implementation using Gibbs sampling
- Document-term matrix creation
- Topic word extraction
- Vocabulary filtering
- Stopword removal
- Uses only built-in libraries

How to Run:
```bash
python EXP8_Simplified.py
```
Expected Output:
- Processing status for each HTML file
- Vocabulary size
- Most common words
- Document-term matrix shape
- LDA fitting progress
- Hidden topics with word probabilities

Common Features Across All Programs

1. No External Dependencies

All programs use only Python standard library modules:
- `csv` - for reading CSV files
- `collections` - for Counter, defaultdict
- `itertools` - for combinations
- `math` - for mathematical operations
- `random` - for random sampling
- `re` - for regular expressions
- `os` - for file operations
- `html.parser` - for HTML parsing

2. Error Handling

- File not found errors
- Empty data handling
- Invalid input validation
- Graceful error messages

3. User-Friendly Output

- Clear progress indicators
- Formatted tables and results
- Interactive testing where applicable
- Detailed statistics and metrics

4. Modular Design

- Separate classes for different functionalities
- Reusable functions
- Clear separation of concerns
- Easy to understand and modify

How to Use During Exam

1. Preparation

- Copy all Python files to your exam directory
- Ensure CSV and HTML files are in the correct locations
- Test each program before the exam

2. Running Programs

- Each program can be run independently
- No installation of external libraries required
- Programs will automatically find and process the data files

3. Understanding the Code

- Each program is well-commented
- Variable names are descriptive
- Functions are small and focused
- Logic is straightforward and easy to follow

4. Modifying Parameters

- Support and confidence thresholds can be adjusted
- Number of topics can be changed
- Minimum word frequencies can be modified
- All parameters are clearly marked in the code

Troubleshooting

Common Issues:

1. **File not found errors**: Check file paths and ensure files exist
2. **Empty results**: Try lowering thresholds or check data quality
3. **Memory issues**: Reduce number of iterations or vocabulary size
4. **Encoding errors**: Ensure files are saved with UTF-8 encoding

Solutions:

1. Verify file paths in the code
2. Check data file formats and content
3. Adjust parameters for your specific dataset
4. Use smaller datasets for testing

Key Algorithms Implemented

1. Apriori Algorithm

- Generates candidate itemsets
- Prunes based on support threshold
- Calculates association rules

2. Naive Bayes

- Calculates prior probabilities
- Computes likelihood with smoothing
- Makes predictions using log probabilities

3. TF-IDF

- Calculates term frequencies

- Computes inverse document frequencies
- Generates TF-IDF scores

4. LDA (Latent Dirichlet Allocation)

- Implements Gibbs sampling
- Updates topic assignments
- Calculates topic-word distributions

5. Sentiment Analysis

- Lexicon-based approach
- Handles negation and intensifiers
- Calculates sentiment scores

Performance Notes

- Programs are optimized for clarity over speed
- Some operations may be slower than optimized libraries
- Suitable for small to medium datasets
- Memory usage is kept minimal

Conclusion

These simplified programs provide all the functionality of the original complex programs while being easy to understand and implement. They use only built-in Python libraries, making them perfect for exam environments without internet access. Each program is self-contained and can be run independently to produce meaningful results.