NLP Concepts

Code 1: Text Preprocessing Pipeline (1.py)

Concept Used:

Text preprocessing pipeline with cleaning, tokenization, stop word removal, and lemmatization.

Observation:

Comprehensive text normalization handling URLs, emails, mentions, and linguistic processing.

Logic of the Code:

- **basic_clean()**: Lowercase conversion, regex cleaning, whitespace normalization
- **tokenize_stop_lemma()**: spaCy processing, lemmatization, stop word filtering
- **preprocess()**: Combines cleaning and tokenization steps

Output Screenshot:

```
['email', 'like', 'filter', 'love', 'nlp', 'visit']
```

Code 2: Sentiment Analysis Pipeline (2.py)

Concept Used:

Binary sentiment classification using TF-IDF vectorization and Logistic Regression pipeline.

Observation:

Complete ML workflow for text classification with balanced dataset and performance evaluation.

Logic of the Code:

- **Dataset**: 10 balanced samples, stratified train-test split
- **Pipeline**: TfidfVectorizer (1,2 n-grams) + LogisticRegression
- **Evaluation**: Classification report, confusion matrix, probability predictions

Output Screenshot:

```
Classification report:
              precision    recall  f1-score   support

           0     0.0000    0.0000    0.0000         2
           1     0.3333    1.0000    0.5000         1

    accuracy                         0.3333         3
   macro avg     0.1667    0.5000    0.2500         3
weighted avg     0.1111    0.3333    0.1667         3

Confusion matrix:
 [[0 2]
 [0 1]]
Predictions: [1 1]
Class probabilities: [[0.42652933 0.57347067]
 [0.46719711 0.53280289]]
```

Code 3: Hyperparameter Optimization (3.py)

Concept Used:

Grid search cross-validation for automated hyperparameter tuning of text classification pipelines.

Observation:

Systematic approach to finding optimal model configurations through exhaustive parameter search.

Logic of the Code:

- **Parameter Grid**: TF-IDF + LogisticRegression parameters (24 combinations)
- **Grid Search**: 3-fold CV, F1-score optimization, parallel processing
- **Model Selection**: Returns best configuration based on CV performance

Output Screenshot:

```
Best params: {'clf__C': 4.0, 'tfidf__analyzer': 'char_wb', 'tfidf__min_df': 1, 'tfidf__ngram_range': (1, 2)}
Best CV score (f1): 0.7222222222222222
Sample prediction: [1]
```

Code 4: Topic Modelling with LDA (4.py)

Concept Used:

Latent Dirichlet Allocation (LDA) for unsupervised topic discovery using bag-of-words representation.

Observation:

Successfully identifies distinct topics (pets vs. finance) from document collection.

Logic of the Code:

- **Preprocessing**: CountVectorizer with stop word removal
- **LDA Training**: 2 topics, learns topic-word distributions
- **Topic Analysis**: Shows top words per topic and document-topic probabilities

Output Screenshot:

```
Topic 0: love dogs play fetch bark rose
Topic 1: investors inflation ease expect sleep purr
Topic distribution: [[0.5 0.5]]
```

Code 5: Named Entity Recognition and POS Tagging (5.py)

Concept Used:

Named Entity Recognition, POS tagging, and syntactic parsing using spaCy.

Observation:

Comprehensive linguistic analysis identifying entities, grammatical roles, and structures.

Logic of the Code:

- **NER**: Identifies organizations, locations, dates, persons
- **POS Tagging**: Extracts parts-of-speech and lemmas
- **Parsing**: Extracts noun chunks and syntactic relationships

Output Screenshot:

```
Named Entities (text, label):
Apple                          -> ORG
Bengaluru                      -> GPE
next quarter                   -> DATE
Tim Cook                       -> PERSON
Karnataka                      -> GPE
September 3, 2025              -> DATE

Part-of-Speech & Lemmas:
Apple           POS=PROPN   Lemma=Apple
is              POS=AUX     Lemma=be
opening         POS=VERB    Lemma=open
a               POS=DET     Lemma=a
new             POS=ADJ     Lemma=new
office          POS=NOUN    Lemma=office
in              POS=ADP     Lemma=in
Bengaluru       POS=PROPN   Lemma=Bengaluru
next            POS=ADJ     Lemma=next
quarter         POS=NOUN    Lemma=quarter
.               POS=PUNCT   Lemma=.
Tim             POS=PROPN   Lemma=Tim
Cook            POS=PROPN   Lemma=Cook
met             POS=VERB    Lemma=meet
Karnataka       POS=PROPN   Lemma=Karnataka
officials       POS=NOUN    Lemma=official
on              POS=ADP     Lemma=on
September       POS=PROPN   Lemma=September
3               POS=NUM     Lemma=3
,               POS=PUNCT   Lemma=,
2025            POS=NUM     Lemma=2025
to              POS=PART    Lemma=to
discuss         POS=VERB    Lemma=discuss
expansion       POS=NOUN    Lemma=expansion
.               POS=PUNCT   Lemma=.

Noun chunks (base NPs):
['Apple',
 'a new office',
 'Bengaluru',
 'Tim Cook',
 'Karnataka officials',
 'September',
 'expansion']
```

Code 6: Document Similarity Search (6.py)

Concept Used:

Information retrieval using TF-IDF vectorization and cosine similarity for document ranking.

Observation:

Effective semantic search system ranking documents by query relevance.

Logic of the Code:

- **Vectorization**: TF-IDF with n-grams and stop word removal
- **Similarity**: Cosine similarity between query and document vectors
- **Search**: Ranks documents by similarity scores, returns top-k results

Output Screenshot:

```
0.344  deep learning methods for image c0.000  transfer learning for NLP0.000  transfer learning for NLP tasks
0.000  classical machine learning with SVM and logistic regression
```

Code 7: Extractive Text Summarization (7.py)

Concept Used:

Frequency-based extractive summarization using term frequency analysis and sentence scoring.

Observation:

Identifies key sentences containing important information while preserving original structure.

Logic of the Code:

- **Segmentation**: Regex-based sentence splitting
- **Frequency Analysis**: Word frequency calculation with stop word removal
- **Sentence Scoring**: Scores based on constituent word frequencies, selects top sentences

Output Screenshot:

```
Transformers have revolutionized natural language processing. By leveraging self-attention, they capture long-range dependencies effectively.
```

Code 8: Bag of Words Model (8.py)

Concept Used:

Bag-of-Words text representation using CountVectorizer for numerical feature conversion.

Observation:

Fundamental text vectorization showing document-to-numerical transformation.

Logic of the Code:

- **Vocabulary Building**: Creates word-to-index mapping from corpus
- **Vectorization**: Converts documents to word count vectors
- **Matrix View**: DataFrame representation showing document-term relationships

Output Screenshot:

```
Vocabulary: ['and' 'are' 'coding' 'fun' 'in' 'is' 'language' 'love' 'natural' 'nlp'
 'powerful' 'processing' 'python']

Bag of Words Matrix:
   and  are  coding  fun  ...  nlp  powerful  processing  python
0    0    0       0    0  ...    0         0           1       0
1    0    0       0    1  ...    0         0           1       0
2    0    0       1    0  ...    0         0           0       1
3    1    1       0    0  ...    1         1           0       1

[4 rows x 13 columns]
PS D:\NLP> python 9.py
Vocabulary: ['advances' 'and' 'artificial' 'deep' 'fun' 'intelligence' 'is' 'learning'
 'machine']

TF-IDF Matrix:
   advances    and  artificial  ...     is  learning  machine
0     0.000  0.000       0.000  ...  0.609     0.360    0.360
1     0.552  0.000       0.000  ...  0.000     0.326    0.326
2     0.000  0.552       0.552  ...  0.000     0.326    0.326

[3 rows x 9 columns]
```

Code 9: TF-IDF Vectorization (9.py)

Concept Used:

TF-IDF weighting scheme emphasizing discriminative terms while reducing common word impact.

Observation:

More nuanced text representation than word counts, highlighting document-specific terms.

Logic of the Code:

- **TF-IDF Calculation**: Balances term frequency with inverse document frequency
- **Normalization**: Unit-length vectors for fair comparison
- **Feature Analysis**: Shows weighted importance of terms per document

Output Screenshot:

```
Vocabulary: ['advances' 'and' 'artificial' 'deep' 'fun' 'intelligence' 'is' 'learning'
 'machine']

TF-IDF Matrix:
   advances    and  artificial  ...     is  learning  machine
0     0.000  0.000       0.000  ...  0.609     0.360    0.360
1     0.552  0.000       0.000  ...  0.000     0.326    0.326
2     0.000  0.552       0.552  ...  0.000     0.326    0.326

[3 rows x 9 columns]
```

Code 10: Word2Vec Embeddings (10.py)

Concept Used:

Word2Vec neural embeddings using Skip-gram architecture for dense word representations.

Observation:

Captures semantic relationships enabling similarity computations and analogical reasoning.

Logic of the Code:

- **Skip-gram Training**: Predicts context words from target word
- **Vector Learning**: 50-dimensional embeddings from co-occurrence patterns
- **Similarity Analysis**: Cosine similarity for semantic word relationships

Output Screenshot:

```
Vector for 'language':
 [ 1.56351421e-02 -1.90203730e-02 -4.11062239e-04  6.93839323e-03
 -1.87794445e-03  1.67635437e-02  1.80215668e-02  1.30730132e-02
 -1.42324204e-03  1.54208085e-02 -1.70686692e-02  6.41421322e-03
 -9.27599426e-03 -1.01779103e-02  7.17923651e-03  1.07406788e-02
  1.55390287e-02 -1.15330126e-02  1.48667218e-02  1.32509926e-02
 -7.41960062e-03 -1.74912829e-02  1.08749345e-02  1.30195115e-02
 -1.57510047e-03 -1.34197120e-02 -1.41718509e-02 -4.99412045e-03
  1.02865072e-02 -7.33047491e-03 -1.87401194e-02  7.65347946e-03
  9.76895820e-03 -1.28571270e-02  2.41711619e-03 -4.14975407e-03
  4.88066689e-05 -1.97670180e-02  5.38400887e-03 -9.50021297e-03
  2.17529293e-03 -3.15244915e-03  4.39334614e-03 -1.57631524e-02
 -5.43436781e-03  5.32639725e-03  1.06933638e-02 -4.78302967e-03
 -1.90201886e-02  9.01175756e-03]

Most similar to 'learning': [('love', 0.21057100594043732), ('deep', 0.16704076528549194), ('natural', 0.150198832154274), ('python', 0.1320440024137497), ('processing', 0.1267007291316986), ('artificial', 0.0
1807), ('is', 0.042373016476631165), ('fun', 0.04067763686180115), ('for', 0.012442179024219513), ('advances', -0.012591069564223329)]

Similarity between 'python' and 'language': 0.044917464
```

Code 11: Naive Bayes Classification (11.py)

Concept Used:

Multinomial Naive Bayes with Bag-of-Words for probabilistic text classification.

Observation:

Effective performance despite strong feature independence assumption.

Logic of the Code:

- **Data Preparation**: Balanced sentiment dataset with train-test split
- **Naive Bayes**: Assumes feature independence, calculates class probabilities
- **Evaluation**: Classification report with precision, recall, F1-scores

Output Screenshot:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       0.00      0.00      0.00         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
 [0.64424596 1.         0.12413287]
 [0.         0.12413287 1.        ]]
```

Code 12: Document Similarity Analysis (12.py)

Concept Used:

Cosine similarity computation between TF-IDF document vectors for semantic similarity measurement.

Observation:

Effectively captures thematic relationships showing high ML/NLP similarity, low cooking similarity.

Logic of the Code:

- **Vector Space**: TF-IDF transforms documents into high-dimensional space
- **Cosine Similarity**: Measures angle between vectors (0-1 scale)
- **Matrix Analysis**: Symmetric similarity matrix for document relationships

Output Screenshot:

```
Cosine Similarity Matrix:
 [[1.         0.64424596 0.        ]
 [0.64424596 1.         0.12413287]
 [0.         0.12413287 1.        ]]
```

Code 13: LDA Topic Modeling with Gensim (13.py)

Concept Used:

LDA topic modeling with Gensim for probabilistic thematic document analysis.

Observation:

Successfully separates technology and cooking topics through statistical analysis.

Logic of the Code:

- **Preprocessing**: Tokenization, dictionary creation, bag-of-words corpus
- **LDA Training**: Learns topic-word and document-topic distributions
- **Topic Analysis**: Displays word probabilities for interpretable topics

Output Screenshot:

```
Topic 0: 0.061*"the" + 0.060*"intelligence" + 0.060*"is" + 0.060*"future" + 0.060*"artificial" + 0.059*"my" + 0.059*"hobbies" + 0.059*"are" + 0.059*"cooking" + 0.059*"baking"
Topic 1: 0.069*"and" + 0.067*"learning" + 0.067*"i" + 0.041*"are" + 0.040*"language" + 0.040*"deep" + 0.040*"processing" + 0.040*"love" + 0.040*"natural" + 0.040*"closely"
```