Handout 2: Implementing LDA for Topic Discovery (Intermediate Level)

Machine Learning for Smarter Innovation

1 Handout 2: Implementing LDA for Topic Discovery (Intermediate Level)

1.1 Understanding Latent Dirichlet Allocation (LDA)

LDA is a probabilistic model that discovers topics by assuming documents are mixtures of topics, and topics are mixtures of words. It's the industry standard for topic modeling.

1.2 How LDA Works

1.2.1 The Generative Story

1. **For each topic**: Define a distribution over words
2. **For each document**:
   - Choose a distribution over topics
   - For each word position:
     - Pick a topic from the document's distribution
     - Pick a word from that topic's distribution

1.2.2 Key Parameters

- **num_topics (K)**: How many topics to find
- **alpha**: Document-topic density (lower = fewer topics per doc)
- **beta/eta**: Topic-word density (lower = fewer words per topic)

1.3 Complete Implementation Guide

1.3.1 Step 1: Data Preparation

```
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download required NLTK data
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
```

```
# Initialize tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    """Clean and prepare text for topic modeling."""
    # Lowercase
    text = text.lower()

    # Remove special characters, keep only letters and spaces
    text = re.sub(r'[^a-z\s]', '', text)

    # Tokenize
    tokens = text.split()

    # Remove stopwords and short words
    tokens = [token for token in tokens
              if token not in stop_words and len(token) > 2]

    # Lemmatize
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    return tokens

# Load your data
df = pd.DataFrame({
    'text': [
        "The new smartphone has amazing battery life and fast charging",
        "I love the design, it's sleek and modern looking",
        "Customer service was helpful when I had issues",
        # ... more documents
    ]
})

# Preprocess all documents
processed_docs = df['text'].apply(preprocess_text).tolist()
```

1.3.2 Step 2: Build LDA Model

```
from gensim import corpora, models
import numpy as np

# Create dictionary and corpus
dictionary = corpora.Dictionary(processed_docs)

# Filter extremes (optional but recommended)
dictionary.filter_extremes(
    no_below=2,       # Ignore words in less than 2 documents
    no_above=0.5,     # Ignore words in more than 50% of documents
    keep_n=1000       # Keep top 1000 most frequent words
)

# Create bag-of-words representation
corpus = [dictionary.doc2bow(doc) for doc in processed_docs]

# Build LDA model
lda_model = models.LdaModel(
    corpus=corpus,
    id2word=dictionary,
    num_topics=5,              # Number of topics
    random_state=42,           # For reproducibility
```

```
    passes=10,                    # Number of passes through corpus
    alpha='auto',                 # Learn optimal alpha
    per_word_topics=True          # Compute word-topic probabilities
)
```

1.3.3 Step 3: Explore Topics

```
# Print topics with top words
def display_topics(model, num_words=10):
    """Display topics with their top words."""
    for idx, topic in model.print_topics(num_words=num_words):
        print(f"\nTopic {idx}:")
        # Parse the topic string
        words = topic.split('+')
        for word in words:
            prob, term = word.split('*')
            term = term.strip().strip('"')
            print(f"  {term}: {float(prob):.3f}")

display_topics(lda_model)

# Get topic distribution for a specific document
doc_topics = lda_model.get_document_topics(corpus[0])
print(f"\nDocument 0 topic distribution:")
for topic_id, prob in doc_topics:
    print(f"  Topic {topic_id}: {prob:.3f}")
```

1.3.4 Step 4: Evaluate Model Quality

```
from gensim.models import CoherenceModel

# Calculate coherence score
coherence_model = CoherenceModel(
    model=lda_model,
    texts=processed_docs,
    dictionary=dictionary,
    coherence='c_v'
)

coherence_score = coherence_model.get_coherence()
print(f"\nCoherence Score: {coherence_score:.3f}")

# Interpretation:
# > 0.5: Good
# 0.4-0.5: Acceptable
# < 0.4: Poor (try different parameters)
```

1.3.5 Step 5: Optimize Number of Topics

```
def find_optimal_topics(corpus, dictionary, texts, min_topics=5, max_topics
    =20):
    """Find optimal number of topics using coherence."""
    coherence_scores = []

    for num_topics in range(min_topics, max_topics + 1):
        model = models.LdaModel(
```

```
            corpus=corpus,
            id2word=dictionary,
            num_topics=num_topics,
            random_state=42,
            passes=10,
            alpha='auto'
        )

        coherence_model = CoherenceModel(
            model=model,
            texts=texts,
            dictionary=dictionary,
            coherence='c_v'
        )

        coherence = coherence_model.get_coherence()
        coherence_scores.append((num_topics, coherence))
        print(f"Topics: {num_topics}, Coherence: {coherence:.3f}")

    # Find best number
    best = max(coherence_scores, key=lambda x: x[1])
    print(f"\nOptimal number of topics: {best[0]} (coherence: {best[1]:.3f})")

    return coherence_scores

# Run optimization
scores = find_optimal_topics(corpus, dictionary, processed_docs)
```

1.3.6 Step 6: Visualize Topics

```
import pyLDAvis
import pyLDAvis.gensim_models as gensimvis

# Create interactive visualization
vis = gensimvis.prepare(lda_model, corpus, dictionary)

# Save as HTML
pyLDAvis.save_html(vis, 'lda_visualization.html')
print("Visualization saved as 'lda_visualization.html'")

# Display in Jupyter notebook
# pyLDAvis.display(vis)
```

1.4 Advanced Techniques

1.4.1 1. Online Learning (for large datasets)

```
# For streaming data or very large corpora
lda_online = models.LdaModel(
    corpus=corpus,
    id2word=dictionary,
    num_topics=10,
    update_every=1,      # Update model every document
    chunksize=100,       # Process 100 documents at a time
    passes=1,            # Single pass for online learning
    alpha='auto'
)
```

1.4.2 2. Domain-Specific Stopwords

```
# Add domain-specific words to filter
domain_stopwords = {'product', 'item', 'thing', 'stuff'}
stop_words.update(domain_stopwords)
```

1.4.3 3. Bigrams and Trigrams

```
from gensim.models import Phrases

# Detect common phrases
bigram = Phrases(processed_docs, min_count=5, threshold=100)
bigram_mod = bigram.freeze()

# Apply to documents
processed_docs_bigrams = [bigram_mod[doc] for doc in processed_docs]
```

1.5 Practical Tips

1.5.1 Preprocessing Best Practices

1. **Keep domain knowledge**: Don't remove important domain terms
2. **Balance filtering**: Too aggressive = loss of meaning
3. **Preserve phrases**: "machine learning" should stay together
4. **Consider POS tagging**: Keep only nouns and verbs

1.5.2 Parameter Tuning

- **Start with defaults**: num_topics=10, alpha='auto', beta='auto'
- **Use coherence**: Not perplexity for evaluation
- **Grid search carefully**: Topics × alpha × beta = many combinations
- **Validate with humans**: Coherence doesn't guarantee usefulness

1.5.3 Common Pitfalls

1. **Too few documents**: Need 100+ per expected topic
2. **Too many topics**: Overfitting, uninterpretable
3. **No preprocessing**: Garbage in, garbage out
4. **Ignoring coherence**: Random topics aren't useful
5. **Not iterating**: First model is rarely the best

1.6 Exercise: Build Your Own Topic Model

1.6.1 Dataset

Use this product review dataset:

```
reviews = [
    "Great battery life, lasts all day",
    "Beautiful design and premium feel",
    "Fast shipping and good packaging",
    # Add 20+ more reviews covering various aspects
]
```

1.6.2 Tasks

1. Preprocess the reviews
2. Build an LDA model with 3-5 topics
3. Calculate coherence score
4. Interpret the topics
5. Find optimal number of topics

1.6.3 Expected Output

- Topic 0: Battery/Power
- Topic 1: Design/Aesthetics
- Topic 2: Shipping/Service
- Coherence > 0.4

1.7 Next Steps

- Try different preprocessing strategies
- Experiment with NMF as alternative
- Apply to your organization's data
- Build a topic-based recommendation system

Remember: Good topic modeling is iterative. Experiment, evaluate, and refine.