Classic Embeddings: LSA & LDA

Lecture 9: The Foundations of Distributional Semantics

PSYC 51.07: Models of Language and Communication - Week 3

Winter 2026

Today's Lecture 📋

1. 🧠 **The Distributional Hypothesis**
2. 📊 **Vector Space Models**
3. 📚 **Latent Semantic Analysis (LSA)**
4. 🎲 **Latent Dirichlet Allocation (LDA)**
5. 🎨 **Modern Topic Modeling: BERTopic**
6. 🔍 **Evaluation & Applications**

Goal: Understand how classic methods represent meaning through co-occurrence

The Fundamental Question 🤔

How do we represent the meaning of words in a way that computers can understand?

\pause

Traditional Approach:

- Dictionaries
- Taxonomies (WordNet)
- Manual feature engineering
- Knowledge bases

Labor-intensive, hard to scale

Distributional Approach:

- Learn from data

The Distributional Hypothesis 📖

"You shall know a word by the company it keeps"

--- J.R. Firth (1957)

\pause

Key Idea:
Words that appear in similar contexts tend to have similar meanings.

Example

- "The **cat** sat on the mat"
- "The **dog** sat on the mat"
- "The **kitten** sat on the mat"

$\rightarrow$ *cat*, *dog*, *kitten* are semantically related

From Words to Vectors 🗺️

Goal: Represent each word as a point in high-dimensional space

Concrete Example
Consider a tiny corpus with 3 documents:

- Doc 1: "The cat sat on the mat"
- Doc 2: "The dog ran in the park"
- Doc 3: "The cat and dog played"

Word-context co-occurrence:

Word	appears with "the"	appears with "sat"	appears with "ran"
cat	2	1	0
dog	2	0	1

Building a Term-Document Matrix 📊

Step 1: Count how often each word appears in each document

cat	0	4	1	0
computer	0	0	0	5
code	0	0	0	3
animal	2	3	2	0

Issues:

- Very sparse (mostly zeros)
- High dimensional (vocab size × doc count)
- Doesn't capture semantic relationships

\pause

TF-IDF Weighting ⚖️

Term Frequency - Inverse Document Frequency

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

Term Frequency (TF):

$$\text{TF}(t, d) = \frac{\text{count}(t, d)}{\sum_{t'} \text{count}(t', d)}$$

How often does term t appear in document d?

Rewards frequent terms in document

Inverse Document Frequency (IDF):

$$\text{IDF}(t) = \log \frac{N}{|\{d : t \in d\}|}$$

TF-IDF: Worked Example 🔢

Corpus: 3 documents, 1000 total documents in collection

Document	Text
Doc 1	"machine learning is great"
Doc 2	"deep learning models"
Doc 3	"machine translation works"

Calculate TF-IDF for "learning" in Doc 1:

```
1Step 1: TF("learning", Doc1) = 1/4 = 0.25  (1 occurrence, 4 words)
2
3Step 2: IDF("learning") = log(1000/500) = log(2) = 0.301
4        (appears in 500 of 1000 docs)
5
```

Latent Semantic Analysis (LSA) 📚

The OG of semantic embeddings (1990)

Algorithm:

1. Build term-document matrix X
2. Apply TF-IDF weighting
3. Perform SVD: $X = U\Sigma V^T$
4. Keep top k dimensions
5. Use U_k as word embeddings

Key Innovation:

- Discovers latent topics
- Solves synonymy problem

LSA: Step-by-Step Worked Example 🔍

Mini corpus (4 words × 3 documents):

```
1           Doc1    Doc2    Doc3
2cat         2       0       1
3dog         0       3       1
4pet         1       2       0
5animal      1       1       1
```

Step 1: Apply SVD $X = U\Sigma V^T$

Step 2: Keep top 2 dimensions (k=2)

```
1from sklearn.decomposition import TruncatedSVD
2import numpy as np
3
4X = np.array([[2,0,1], [0,3,1], [1,2,0], [1,1,1]])
```

LSA: How It Works 🔍

Example: Discovering latent topics

Original Dimensions:

- 50,000 terms
- 10,000 documents
- Matrix: 50k × 10k
- Sparse and noisy

After LSA:

- 100-300 latent dimensions
- Dense representation
- Captures semantic patterns

LSA Limitations ⚠️

Computational Issues:

- SVD is expensive: $O(min(nm^2, n^2m))$
- Hard to update with new documents
- Memory intensive for large corpora

Modeling Issues:

- Linear relationships only
- Bag-of-words (loses word order)
- No polysemy handling
- Negative values (hard to interpret)

Example Problem: Polysemy

Latent Dirichlet Allocation (LDA) 🎲

A probabilistic approach to topic modeling

What if we model documents as probabilistic mixtures of topics?

Key Assumptions:

1. Each document is a mixture of topics
2. Each topic is a distribution over words
3. The mixture proportions vary per document

Advantages over LSA:

- Probabilistic interpretation
- Non-negative weights
- Better for topic discovery

LDA: The Generative Story 📖

How LDA imagines documents are created:

Generative Process:

1. Choose number of topics K
2. For each topic k:
    - Draw word distribution $\phi_k \sim \text{Dir}(\beta)$
3. For each document d:
    - Draw topic distribution $\theta_d \sim \text{Dir}(\alpha)$
    - For each word position n:
        - Choose topic $z_{dn} \sim \text{Mult}(\theta_d)$
        - Choose word $w_{dn} \sim \text{Mult}(\phi_{z_{dn}})$

Key Parameters:

LDA: Concrete Generative Example 🎲

Imagine generating a document about "tech pets":

```
Step 1: Pick topic mixture for this document
        θ_doc = [0.6 Tech, 0.3 Animals, 0.1 Sports]

Step 2: For each word, sample a topic, then sample a word:

Word 1: Sample topic → Tech (60% chance)
        Sample word from Tech → "software"

Word 2: Sample topic → Animals (30% chance)
         Sample word from Animals → "cat"

Word 3: Sample topic → Tech (60% chance)
        Sample word from Tech → "computer"

Word 4: Sample topic → Animals (30% chance)
        Sample word from Animals → "pet"
```

LDA in Practice 💻

```
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer

# Prepare documents
documents = [
    "The cat sat on the mat",
    "The dog ran in the park",
    "Machine learning is amazing",
    # ... more documents
]

# Create bag-of-words representation
vectorizer = CountVectorizer(max_features=1000, stop_words='english')
doc_term_matrix = vectorizer.fit_transform(documents)
vocab = vectorizer.get_feature_names_out()

# Fit LDA model
lda = LatentDirichletAllocation(
```

LDA in Practice 💻

```
21    max_iter=50
22)
23
24doc_topics = lda.fit_transform(doc_term_matrix)
25
26# Print top words per topic
27for idx, topic in enumerate(lda.components_):
28    top_words = [vocab[i] for i in topic.argsort()[-10:]]
29    print(f"Topic {idx}: {', '.join(top_words)}")
```

...continued

LDA Example: Topic Discovery 🔍

Example Topics from News Corpus:

Topic 1: Politics (20%)

```
government, election, president, vote, policy, senate, congress, party
```

Topic 2: Sports (15%)

```
game, team, player, score, win, season, coach, championship
```

Topic 3: Technology (25%)

```
software, computer, data, algorithm, system, internet, code, AI
```

Topic 4: Finance (40%)

```
market, stock, price, investment, economy, trade, bank, profit
```

Document representation: [0.20, 0.15, 0.25, 0.40] ← mainly finance

Modern Topic Modeling: BERTopic 🎨

Combining neural embeddings with topic models

BERTopic Pipeline:

1. Embed documents with BERT
2. Reduce dimensionality with UMAP
3. Cluster with HDBSCAN
4. Generate topics via c-TF-IDF
5. Fine-tune with MaximalMarginalRelevance

Advantages:

- Leverages pre-trained models
- Better semantic understanding
- Automatic topic number detection

Modern Topic Modeling: BERTopic 🎨

```
21# Get topic info
22topic_info = model.get_topic_info()
23
24# Visualize
25fig = model.visualize_topics()
```

...continued

Reference: Grootendorst (2022) - "BERTopic: Neural topic modeling with a class-based TF-IDF procedure"

LSA vs. LDA vs. BERTopic 📊

Interpretability	Medium	High	High
Topic coherence	Medium	High	Very High
Computation	Fast	Medium	Slow
Scalability	Good	Good	Medium
Pre-training	No	No	Yes
Hyperparameters	Few	Many	Many
Topic number	Fixed	Fixed	Automatic

Recommendations

- **LSA**: Quick exploration, large-scale retrieval
- **LDA**: Interpretable topics, when you know topic count

Evaluating Topic Models 📏

Intrinsic Metrics:

- **Perplexity:** Lower is better
  - Measures likelihood
  - Can be misleading!
- **Topic Coherence:** Higher is better
  - Measures semantic similarity
  - Better correlation with human judgment
  - CV, UCI, UMass variants
- **Topic Diversity:**
  - Unique words across topics
  - Avoids redundant topics

Coherence Evaluation: Code Example 💻

```
from gensim.models import LdaModel
from gensim.models.coherencemodel import CoherenceModel
from gensim.corpora import Dictionary

# Prepare corpus
texts = [doc.split() for doc in documents]
dictionary = Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]

# Train LDA with different topic numbers
coherence_scores = []
for num_topics in [5, 10, 15, 20, 25]:
    lda = LdaModel(corpus, num_topics=num_topics,
                   id2word=dictionary, passes=10)

    # Calculate coherence (C_V is recommended)
    coherence = CoherenceModel(model=lda, texts=texts,
                               dictionary=dictionary,
```

Coherence Evaluation: Code Example 💻

```
21
22# Results: [(5, 0.42), (10, 0.51), (15, 0.48), (20, 0.45), (25, 0.41)]
23# Best: 10 topics with coherence 0.51
```

...continued

Rule of thumb: Higher coherence = more interpretable topics

Applications of Classic Embeddings 🚀

1. **Information Retrieval**
    - Semantic search
    - Document similarity
    - Query expansion
2. **Document Organization**
    - Clustering
    - Topic discovery
    - Trend analysis
3. **Text Mining**
    - Opinion mining
    - Literature review

Application Example: Semantic Search with LSA 🔎

```
1from sklearn.decomposition import TruncatedSVD
2from sklearn.metrics.pairwise import cosine_similarity
3
4# Documents already vectorized with TF-IDF
5# tfidf_matrix shape: (1000 docs, 5000 words)
6
7# Apply LSA to reduce dimensions
8lsa = TruncatedSVD(n_components=100)
9doc_embeddings = lsa.fit_transform(tfidf_matrix)
10
11# User query: "machine learning algorithms"
12query_tfidf = vectorizer.transform(["machine learning algorithms"])
13query_embedding = lsa.transform(query_tfidf)
14
15# Find most similar documents
16similarities = cosine_similarity(query_embedding, doc_embeddings)[0]
17top_5_docs = similarities.argsort()[-5:][::-1]
18
```

Application Example: Semantic Search with LSA 🔎

```
21    print(f"  Doc {idx}: similarity = {similarities[idx]:.3f}")
```

...continued

Key insight: LSA finds documents about "neural networks" and "deep learning" even though those exact words weren't in the query!

Discussion Question 💬

Can distributional semantics truly capture meaning?

Arguments For:

- Captures semantic similarity
- Works well in practice
- Aligns with linguistic theory
- Scalable to large corpora
- Unsupervised learning

Arguments Against:

- Text-only (no grounding)
- Missing embodied experience

Practical Tips 💡

1. **Preprocessing Matters:**
    - Remove stopwords (but not always!)
    - Lemmatization vs. stemming
    - Handle punctuation carefully
    - Consider bigrams/trigrams
2. **Choosing Parameters:**
    - Start with 10-50 topics for LDA
    - Use perplexity for model selection
    - Validate with coherence scores
    - Try multiple random seeds
3. **Interpreting Results:**

Summary 🎯

What we learned today:

1. **Distributional Hypothesis:** Words in similar contexts have similar meanings
2. **LSA (1990):**
    - SVD on term-document matrix
    - Linear dimensionality reduction
    - Fast but limited interpretability
3. **LDA (2003):**
    - Probabilistic topic modeling
    - Documents as mixtures of topics
    - Highly interpretable

Key References 📚

Classic Papers:

- Firth, J.R. (1957). "A synopsis of linguistic theory 1930-1955"
- Deerwester et al. (1990). "Indexing by Latent Semantic Analysis"
- Blei et al. (2003). "Latent Dirichlet Allocation"
- Harnad, S. (1990). "The symbol grounding problem"

Modern Extensions:

- Boleda, G. (2020). "Distributional Semantics and Linguistic Theory"
- Grootendorst, M. (2022). "BERTopic: Neural topic modeling with a class-based TF-IDF procedure"
- Grand et al. (2022). "Semantic projection recovers rich human knowledge of multiple object features"

Questions? 🙋

Next Lecture:

Word Embeddings: Word2Vec, GloVe, FastText

From count-based to prediction-based methods!