Lecture 10: X-Hour Embeddings Workshop

Week 3: Hands-On Dimensionality Reduction and Word Vectors

PSYC 51.07: Models of Language and Communication

Learning Objectives

By the end of this session, you will:

1. Implement classic dimensionality reduction (LSA, LDA)
2. Train and analyze Word2Vec embeddings
3. Visualize high-dimensional embeddings using UMAP
4. Compare different embedding methods on real data
5. Understand semantic relationships captured by embeddings

Workshop format: Hands-on coding with the 20 Newsgroups dataset

Workshop Overview

Today's Agenda:

1. **Part 1:** Why embeddings? From sparse to dense representations
2. **Part 2:** LSA - Latent Semantic Analysis with SVD
3. **Part 3:** LDA - Latent Dirichlet Allocation for topic modeling
4. **Part 4:** Word2Vec - Neural word embeddings
5. **Part 5:** Visualizing embeddings with UMAP
6. **Part 6:** Comparing methods and document classification

Companion notebook: `xhour_embeddings_demo.ipynb`

Part 1: Why Embeddings?

The problem with sparse representations:

Last week (BoW, TF-IDF):

- High dimensional (vocab size)
- Sparse (mostly zeros)
- No semantic similarity
- "dog" and "puppy" are orthogonal

Embeddings:

- Low dimensional (50-300 dims)
- Dense (all non-zero)
- Similar words cluster together

Sparse vs Dense: Concrete Comparison

```
# Sparse representation (one-hot / BoW)
# Vocabulary: [cat, dog, puppy, car, truck, vehicle]

cat_sparse   = [1, 0, 0, 0, 0, 0]  # 6 dimensions, 5 zeros
dog_sparse   = [0, 1, 0, 0, 0, 0]
puppy_sparse = [0, 0, 1, 0, 0, 0]

# Cosine similarity: cat-dog = 0, dog-puppy = 0  (orthogonal!)

# Dense embedding (learned from data)
cat_dense   = [0.8, -0.2, 0.5]   # 3 dimensions, all non-zero
dog_dense   = [0.7, -0.1, 0.6]   # Similar to cat!
puppy_dense = [0.75, -0.15, 0.55]  # Very similar to dog!
car_dense   = [-0.3, 0.9, -0.4]  # Different cluster

# Cosine similarity: cat-dog = 0.98, dog-puppy = 0.99
```


Key insight: Dense embeddings capture that "dog" and "puppy" are semantically related

The Magic of Word Vectors

Famous example: king - man + woman = queen

Vector Arithmetic

Word embeddings capture semantic relationships as directions in space:

- Gender direction: woman - man
- Royalty direction: king - queen
- Pluralization: words - word

Key insight: Meaning encoded as geometry!

Vector Arithmetic: Step-by-Step Example

```
import numpy as np

# Pretend embeddings (simplified to 3D for illustration)
embeddings = {
    'king':   np.array([0.9, 0.8, 0.2]),
    'queen':  np.array([0.85, 0.75, 0.7]),
    'man':    np.array([0.7, 0.6, 0.1]),
    'woman':  np.array([0.65, 0.55, 0.6]),
}

# The analogy: king - man + woman = ?
result = embeddings['king'] - embeddings['man'] + embeddings['woman']
# result = [0.9-0.7+0.65, 0.8-0.6+0.55, 0.2-0.1+0.6]
#        = [0.85, 0.75, 0.7]  ← Very close to 'queen'!

# Why does this work?
# king - man   = "royalty" direction = [0.2, 0.2, 0.1]
# woman + royalty = queen
```

Part 2: Latent Semantic Analysis (LSA)

Using SVD to find latent topics:

$$X \approx U_k \Sigma_k V_k^T$$

Algorithm:

1. Build TF-IDF matrix X
2. Apply Singular Value Decomposition
3. Keep top k dimensions
4. Use U_k as word embeddings

Interpretation:

- U: word-topic associations
- Σ: topic strengths

LSA in Code

```
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer

# Build TF-IDF matrix
tfidf = TfidfVectorizer(max_features=5000, stop_words='english')
tfidf_matrix = tfidf.fit_transform(documents)

# Apply LSA
lsa = TruncatedSVD(n_components=100, random_state=42)
doc_embeddings = lsa.fit_transform(tfidf_matrix)
word_embeddings = lsa.components_.T

print(f"Explained variance: {lsa.explained_variance_ratio_.sum():.2%}")
```

Try it: Find similar words using cosine similarity!

LSA: Finding Similar Words

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Get vocabulary mapping
vocab = tfidf.get_feature_names_out()
word_to_idx = {word: i for i, word in enumerate(vocab)}

def find_similar_words(word, top_n=5):
    """Find words with similar LSA embeddings."""
    if word not in word_to_idx:
        return f"'{word}' not in vocabulary"

    idx = word_to_idx[word]
    word_vec = word_embeddings[idx].reshape(1, -1)

    # Compute similarities to all words
    sims = cosine_similarity(word_vec, word_embeddings)[0]

```

LSA: Finding Similar Words

```
    return [(vocab[i], f"{sims[i]:.3f}") for i in top_indices]

print(find_similar_words("computer"))
# Output: [('software', 0.82), ('program', 0.79),
#          ('system', 0.71), ('hardware', 0.68), ('disk', 0.65)]
```

...continued

Part 3: LDA for Topic Modeling

A probabilistic approach:

Generative Story

LDA imagines documents are created by:

1. Choosing a mixture of topics
2. For each word, picking a topic
3. Sampling a word from that topic

Key difference from LSA:

- Probabilistic interpretation
- Non-negative weights
- More interpretable topics

LDA Example Output

```
Topic 0: hockey, game, team, player, season, nhl, play
Topic 1: space, nasa, launch, orbit, shuttle, satellite
Topic 2: computer, software, program, file, windows, system
Topic 3: medical, doctor, patient, disease, health, treatment
Topic 4: government, president, congress, law, political
```

Each document is a **mixture** of topics:

Document #42: 60% Space + 25% Computer + 15% Other

LDA: Complete Working Example

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

# 20 Newsgroups sample documents
documents = [
    "The hockey team scored three goals in the game",
    "NASA launched a new satellite into orbit",
    "Install the software program on your computer",
    "The doctor prescribed medicine for the patient",
    # ... more documents
]

# Step 1: Create bag-of-words matrix
vectorizer = CountVectorizer(max_features=1000, stop_words='english')
bow_matrix = vectorizer.fit_transform(documents)

# Step 2: Fit LDA
lda = LatentDirichletAllocation(n_components=5, random_state=42)
```

LDA: Complete Working Example

```
# Step 3: Print topics
vocab = vectorizer.get_feature_names_out()
for topic_idx, topic in enumerate(lda.components_):
    top_words = [vocab[i] for i in topic.argsort()[-7:]]
    print(f"Topic {topic_idx}: {', '.join(top_words)}")
```

...continued

Part 4: Word2Vec

Learning embeddings from context:

Skip-gram:
Given target word, predict context

"The **cat** sat on mat"

- cat → the, sat, on

CBOW:
Given context, predict target

the, sat, on → **cat**

```
from gensim.models import Word2Vec

model = Word2Vec(
```

Word2Vec: Semantic Similarity

```
# Find similar words
model.wv.most_similar('computer', topn=5)
# [('software', 0.82), ('program', 0.79), ('system', 0.75), ...]

# Word analogies
model.wv.most_similar(
    positive=['woman', 'king'],
    negative=['man']
)
# [('queen', 0.71), ...]
```

Hands-on Exercise

Try creating your own word analogies! What works? What fails?

Word2Vec: Exploring Analogies

```
# Analogies that typically WORK well:
model.wv.most_similar(positive=['paris', 'germany'], negative=['france'])
# → 'berlin' (capital cities)

model.wv.most_similar(positive=['walking', 'swam'], negative=['swimming'])
# → 'walked' (verb tenses)

model.wv.most_similar(positive=['bigger', 'cold'], negative=['big'])
# → 'colder' (comparatives)

# Analogies that often FAIL:
model.wv.most_similar(positive=['doctor', 'woman'], negative=['man'])
# → might return 'nurse' instead of 'doctor' (reflects bias!)

model.wv.most_similar(positive=['sushi', 'italy'], negative=['japan'])
# → uncertain results (cultural associations are noisy)
```


Discussion: Why do some analogies work better than others?

Part 5: Visualizing with UMAP

Projecting 100D → 2D:

```
import umap

reducer = umap.UMAP(
    n_neighbors=15,
    min_dist=0.1,
    metric='cosine'
)

embeddings_2d = reducer.fit_transform(word_vectors)
```

UMAP advantages:

- Faster than t-SNE
- Preserves global structure

What You Should See

When you visualize embeddings:

Sports cluster:

- hockey, baseball, player, team, game

Space cluster:

- nasa, shuttle, orbit, launch, space

Tech cluster:

- computer, software, program, windows

Medical cluster:

- doctor, patient, hospital, treatment

Related words should cluster together even though we never told the model they were

UMAP Visualization: Complete Code

```
import umap
import matplotlib.pyplot as plt

# Get word vectors for a subset of interesting words
words_to_plot = ['hockey', 'baseball', 'player', 'team', 'game',
                 'nasa', 'shuttle', 'orbit', 'space', 'satellite',
                 'computer', 'software', 'program', 'windows', 'disk',
                 'doctor', 'patient', 'hospital', 'disease', 'treatment']

word_vectors = np.array([model.wv[w] for w in words_to_plot])

# Reduce to 2D with UMAP
reducer = umap.UMAP(n_neighbors=5, min_dist=0.3, metric='cosine')
embeddings_2d = reducer.fit_transform(word_vectors)

# Plot
plt.figure(figsize=(12, 8))
plt.scatter(embeddings_2d[:, 0], embeddings_2d[:, 1], alpha=0.7)
```

UMAP Visualization: Complete Code

```
plt.title("Word Embeddings Visualized with UMAP")
plt.savefig("word_clusters.png")
```

...continued

Part 6: Comparing Methods

Method	Speed	Interpretability	Quality	Data Needed
LSA	Fast	Medium	Medium	Small-Medium
LDA	Medium	High	Medium	Medium
Word2Vec	Medium	Low	High	Large

Recommendations:

- **Quick exploration:** LSA
- **Interpretable topics:** LDA
- **Best semantic quality:** Word2Vec

Document Classification with Embeddings

Using embeddings as features:

```
def document_vector(doc, model):
    """Average word vectors for document."""
    tokens = preprocess(doc)
    vectors = [model.wv[w] for w in tokens if w in model.wv]
    return np.mean(vectors, axis=0) if vectors else np.zeros(100)

# Train classifier
X_train = [document_vector(doc, w2v) for doc in train_docs]
clf = LogisticRegression()
clf.fit(X_train, y_train)
```

Compare to TF-IDF baseline!

Classification Comparison: Full Example

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

# Method 1: TF-IDF baseline
tfidf = TfidfVectorizer(max_features=5000)
X_tfidf = tfidf.fit_transform(train_docs)
clf_tfidf = LogisticRegression(max_iter=1000)
tfidf_scores = cross_val_score(clf_tfidf, X_tfidf, y_train, cv=5)

# Method 2: LSA embeddings
lsa = TruncatedSVD(n_components=100)
X_lsa = lsa.fit_transform(X_tfidf)
clf_lsa = LogisticRegression(max_iter=1000)
lsa_scores = cross_val_score(clf_lsa, X_lsa, y_train, cv=5)

# Method 3: Word2Vec embeddings
X_w2v = np.array([document_vector(doc, model) for doc in train_docs])
clf_w2v = LogisticRegression(max_iter=1000)
```

Classification Comparison: Full Example

```
print(f"TF-IDF:   {tfidf_scores.mean():.3f} (+/- {tfidf_scores.std():.3f})")
print(f"LSA:      {lsa_scores.mean():.3f} (+/- {lsa_scores.std():.3f})")
print(f"Word2Vec: {w2v_scores.mean():.3f} (+/- {w2v_scores.std():.3f})")
```

...continued

Key Takeaways

1. **Embeddings capture semantic meaning** - similar words have similar vectors
2. **Different methods, different strengths:**
    - LSA: Fast, linear, interpretable
    - LDA: Probabilistic, topic-focused
    - Word2Vec: Neural, best for similarity
3. **Visualization reveals structure** - UMAP shows semantic clusters
4. **Limitations:**
    - Static (one vector per word, no context)
    - Requires substantial data
    - Can encode biases

Discussion Questions

1. **Why does vector arithmetic work?** What does "king - man + woman" really mean geometrically?
2. **Bias in embeddings:** If Word2Vec learns from news articles, what biases might it capture?
3. **Window size matters:** What happens with window=2 vs window=10?
4. **Out-of-vocabulary problem:** How do you handle words not in your vocabulary?
5. **When to use what:** For a sentiment analysis task, would you choose LSA, LDA, or Word2Vec?

Next Steps

For Assignment 2:

- Use embeddings to improve your classifier
- Compare at least 2 embedding methods
- Visualize your embeddings

Coming up in Lecture 11:

- Modern neural word embeddings
- GloVe and FastText
- Subword tokenization

Office hours: Available if you need help with the notebook!