Module 9: Text and Embeddings

Overview

Text data requires special treatment for machine learning. Unlike numbers, words have no inherent numerical representation, and the order and context of words matters for meaning. This module progresses from simple count-based representations (bag-of-words, TF-IDF) to learned embeddings that capture semantic relationships. These techniques enable text classification, similarity computation, and information retrieval.

---

1. The Text Representation Challenge

Why Text is Different

- Words are symbolic, not numeric
- Vocabulary can be enormous (millions of words)
- Order matters: “dog bites man” ≠ “man bites dog”
- Meaning is contextual: “bank” (river vs. financial)

The Goal

Convert text to fixed-length numerical vectors that: - Can be used by standard ML algorithms - Capture relevant meaning - Handle variable-length documents

---

2. Bag-of-Words (BoW)

The Idea

Represent each document as a vector of word counts, ignoring order.

Process

1. Build vocabulary from all documents
2. For each document, count occurrences of each vocabulary word
3. Result: vector of length |vocabulary|

Example

Documents: - “the cat sat” - “the dog sat”

Vocabulary: {cat, dog, sat, the}

Vectors: - “the cat sat” → [1, 0, 1, 1] - “the dog sat” → [0, 1, 1, 1]

Limitations

- **Sparsity**: Most words absent from each document
- **No semantics**: “happy” and “joyful” are unrelated
- **No order**: Loses phrase structure
- **Common words dominate**: “the”, “is”, “and”

---

3. TF-IDF

The Problem with Raw Counts

Common words (“the”, “and”) have high counts but carry little meaning. Rare words may be more informative.

Term Frequency (TF)

How often a term appears in this document:

```
TF(t, d) = count(t in d) / |d|
```

Inverse Document Frequency (IDF)

How rare is the term across documents:

```
IDF(t) = log(N / DF(t))
```

Where N = total documents, DF(t) = documents containing t.

TF-IDF Score

```
TF-IDF(t, d) = TF(t, d) × IDF(t)
```

High score = term is frequent in this document but rare overall.

Benefits

- Downweights common words automatically
- Highlights distinctive terms
- Standard baseline for text classification

---

4. N-grams

Beyond Single Words

N-grams are sequences of n consecutive tokens: - Unigrams (1): “the”, “cat”, “sat” - Bigrams (2): “the cat”, “cat sat” - Trigrams (3): “the cat sat”

Why Use N-grams?

- Capture some word order
- Handle phrases: “not good” ≠ “good”
- Improve classification accuracy

Tradeoff

More n-grams = larger vocabulary = more sparsity

---

5. Word Embeddings

The Revolution

Instead of sparse, discrete vectors, represent words as dense, continuous vectors (~100-300 dimensions) learned from data.

Key Properties

- **Semantic similarity**: Similar words have similar vectors
- **Relationships captured**: king - man + woman ≈ queen
- **Learned from context**: Words appearing in similar contexts have similar embeddings

Cosine Similarity

Measure similarity between embedding vectors:

```
cos(A, B) = (A · B) / (||A|| × ||B||)
```

Range: -1 to 1 (1 = identical direction)

---

6. Word2Vec

Skip-Gram Model

Given a word, predict surrounding context words.

Training objective: maximize probability of context words given center word.

CBOW (Continuous Bag of Words)

Given context words, predict center word.

How It Learns Semantics

Words appearing in similar contexts (e.g., “quick” and “fast”) get pushed toward similar vectors during training.

Example Relationships

```
vector("king") - vector("man") + vector("woman")  vector("queen")
vector("paris") - vector("france") + vector("germany")  vector("berlin")
```

---

7. GloVe and FastText

GloVe (Global Vectors)

- Uses co-occurrence statistics
- Learns from word-word co-occurrence matrix
- Often similar quality to Word2Vec

FastText

- Learns embeddings for character n-grams
- Can represent out-of-vocabulary words
- Better for morphologically rich languages
- Made by Facebook Research

---

8. Document Embeddings

From Words to Documents

How to represent a whole document with embeddings?

Simple Averaging

Average the embeddings of all words in the document:

```
doc_vector = (1/n) × Σ word_embeddings
```

Surprisingly effective for many tasks!

TF-IDF Weighted Average

Weight word embeddings by TF-IDF scores before averaging.

Sentence Transformers

Modern models (BERT, Sentence-BERT) produce embeddings for entire sentences/documents directly.

---

9. Transfer Learning for Text

Using Pre-trained Embeddings

1. Download pre-trained embeddings (Word2Vec, GloVe)
2. Look up embeddings for your vocabulary
3. Use as features for your task

This transfers knowledge from massive text corpora to your specific problem.

Benefits

- Works with small labeled datasets
- Captures general language understanding
- Much faster than training embeddings from scratch

---

10. Practical Text Pipeline

Standard Workflow

1. **Preprocessing**: Lowercase, remove punctuation, tokenize
2. **Vectorization**: BoW, TF-IDF, or embeddings
3. **Modeling**: Classic ML or neural network
4. **Evaluation**: Accuracy, F1, etc.

scikit-learn Approach

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

vectorizer = TfidfVectorizer(max_features=5000)
X_train = vectorizer.fit_transform(train_texts)
X_test = vectorizer.transform(test_texts)

model = LogisticRegression()
model.fit(X_train, train_labels)
```

Key Takeaways

1. **Bag-of-words** represents documents as word count vectors
2. **TF-IDF** weights terms by importance (frequent locally, rare globally)
3. **N-grams** capture word order partially
4. **Word embeddings** are dense vectors capturing semantics
5. **Word2Vec** learns embeddings from context prediction
6. **Similar words have similar embeddings**
7. **Document embeddings** can be simple averages of word embeddings

8. **Pre-trained embeddings** enable transfer learning

Connections to Future Modules

- **Module 13**: Transformers and attention (BERT, GPT)

Further Reading

- [Links to be added]
- Word2Vec paper: “Efficient Estimation of Word Representations”
- GloVe paper: “GloVe: Global Vectors for Word Representation”