Lecture 8 超基础复习笔记 - 词嵌入（Word Embeddings）

零、先理解问题（什么是词嵌入？）

传统方法的问题

One-hot编码：

```
词汇表：['cat', 'dog', 'king', 'queen']

cat   = [1, 0, 0, 0]
dog   = [0, 1, 0, 0]
king  = [0, 0, 1, 0]
queen = [0, 0, 0, 1]
```

问题：

```
1. 维度太高
   10万个词 → 10万维向量

2. 无法表示语义关系
   cat和dog都是动物，但向量完全不相关
   king和queen都是royalty，但向量也完全不相关

3. 稀疏性
   向量中只有一个1，其余都是0
```

词嵌入的解决方案

定义：将词映射到低维稠密向量

例子：

```
# 3维词嵌入（实际通常100-300维）
cat   = [0.2, 0.8, 0.1]  ← 动物特征强
dog   = [0.3, 0.7, 0.2]  ← 也是动物，向量相似
king  = [0.9, 0.1, 0.8]  ← royalty特征
queen = [0.8, 0.2, 0.9]  ← 也是royalty，向量相似
```

优点：

```
1. 低维度
   10万个词 → 300维向量（减少333倍）

2. 语义相似
   相似的词有相似的向量
   cat ≈ dog（都是动物）
   king ≈ queen（都是royalty）

3. 稠密
   向量中大部分元素都有意义
```

一、Word2Vec原理

1. 核心思想

分布假说（Distributional Hypothesis）：

```
"You shall know a word by the company it keeps"
（通过一个词的邻居来了解它）

例子：
"The cat sat on the mat"
"The dog sat on the rug"
```

cat和dog出现在相似的上下文中
→ 它们应该有相似的意义
→ 它们应该有相似的向量

---

2. Word2Vec的两种模型

CBOW（Continuous Bag of Words）

目标：用上下文预测中心词

```
输入：上下文词
输出：中心词

例子：
上下文：["The", "cat", "on", "the", "mat"]
预测：sat

上下文：["The", "dog", "on", "the", "rug"]
预测：sat
```

Skip-gram

目标：用中心词预测上下文

```
输入：中心词
输出：上下文词

例子：
输入：sat
预测：["The", "cat", "on", "the", "mat"]

输入：sat
预测：["The", "dog", "on", "the", "rug"]
```

3. 训练过程（简化）

```
步骤1：随机初始化词向量
cat   = [random, random, random]
dog   = [random, random, random]
...

步骤2：遍历训练文本
对于句子："The cat sat on the mat"
  对于每个词及其上下文
    调整向量，使得：
    - 上下文中出现的词，向量更接近
    - 上下文中不出现的词，向量更远离

步骤3：迭代多次，直到收敛

结果：
cat和dog的向量变得相似（因为上下文相似）
```

二、向量运算

1. 余弦相似度（Cosine Similarity）

公式：

```
cos(θ) = (A · B) / (|A| × |B|)

其中：
A · B = A和B的点积
|A| = A的模长
|B| = B的模长
```

手工计算示例：

```
A = [1, 2, 3]
B = [4, 5, 6]

A · B = 1×4 + 2×5 + 3×6 = 4 + 10 + 18 = 32

|A| = √(1² + 2² + 3²) = √14 ≈ 3.74
|B| = √(4² + 5² + 6²) = √77 ≈ 8.77

cos(θ) = 32 / (3.74 × 8.77) = 32 / 32.8 ≈ 0.976
```

解释：

```
cos(θ) ≈ 1  → 向量非常相似（夹角接近0°）
cos(θ) ≈ 0  → 向量不相关（夹角接近90°）
cos(θ) ≈ -1 → 向量相反（夹角接近180°）
```

代码实现：

```
import numpy as np

def cosine_similarity(v1, v2):
    """
    计算两个向量的余弦相似度
    """
    # 点积
    dot_product = np.dot(v1, v2)

    # 模长
    norm_v1 = np.linalg.norm(v1)
    norm_v2 = np.linalg.norm(v2)

    # 余弦相似度
    similarity = dot_product / (norm_v1 * norm_v2)

    return similarity

# 例子
v1 = np.array([1, 2, 3])
```

```
v2 = np.array([4, 5, 6])
print(cosine_similarity(v1, v2))  # 0.974
```

---

2. 语义向量运算

著名例子：

```
king - man + woman ≈ queen
```

为什么可行?

```
king的向量包含:
  - royalty (王室)
  - male (男性)
  - power (权力)

man的向量包含:
  - male (男性)
  - human (人类)

woman的向量包含:
  - female (女性)
  - human (人类)

king - man: 去除"男性"特征, 保留"王室+权力"
king - man + woman: 加上"女性"特征
结果: royalty + female = queen ✓
```

其他例子：

```
Paris - France + Italy ≈ Rome
 (首都关系)

bigger - big + small ≈ smaller
 (比较级关系)
```

```
walking - walk + swim ≈ swimming
（动名词关系）
```

三、使用Gensim训练Word2Vec

1. 准备数据

```
import nltk
from nltk.corpus import gutenberg

# 下载Moby Dick
nltk.download('gutenberg')

# 读取文本
moby_dick = gutenberg.sents('melville-moby_dick.txt')

# 数据格式：
# [['Call', 'me', 'Ishmael', '.'],
#  ['Some', 'years', 'ago', ...],
#  ...]
```

---

2. 训练模型

```
from gensim.models import Word2Vec

# 训练模型（一行代码！）
model = Word2Vec(
    sentences=moby_dick,  # 训练数据
    vector_size=100,      # 向量维度
    window=5,             # 上下文窗口大小
    min_count=5,          # 最小词频（低于此值的词被忽略）
```

```
    workers=4              # 并行线程数
)
```

参数解释：

```
vector_size=100:
  每个词用100维向量表示

window=5:
  考虑前后5个词作为上下文
  例如："The cat sat on the mat"
  对于"sat"，上下文是["The", "cat", "on", "the", "mat"]

min_count=5:
  只训练出现5次以上的词
  过滤掉生僻词和拼写错误

workers=4:
  使用4个CPU核心并行训练
  加速训练过程
```

---

3. 使用模型

获取词向量

```
# 获取单词的向量
whale_vec = model.wv['whale']
print(whale_vec.shape)  # (100,)
print(whale_vec[:5])    # [0.123, -0.456, 0.789, ...]
```

查找相似词

```
# 找最相似的10个词
similar_words = model.wv.most_similar('whale', topn=10)
```

```
print(similar_words)

# 输出示例:
# [('ship', 0.95),
#  ('boat', 0.93),
#  ('sea', 0.91),
#  ('ocean', 0.89),
#  ...]
```

词语类比

```
# king - man + woman ≈ queen
result = model.wv.most_similar(
    positive=['king', 'woman'],  # 加上这些词
    negative=['man'],            # 减去这些词
    topn=1
)
print(result)  # [('queen', 0.85)]
```

---

4. 实现most_similar方法

任务：自己实现类似Gensim的most_similar功能

```
def most_similar(word, model, topn=10):
    """
    找出与给定词最相似的topn个词

    参数:
    word: 目标词
    model: 训练好的Word2Vec模型
    topn: 返回前n个结果

    返回:
    [(word1, similarity1), (word2, similarity2), ...]
    """
```

```
    # 获取目标词的向量
    target_vec = model.wv[word]

    # 存储所有词的相似度
    similarities = []

    # 遍历词汇表中的所有词
    for vocab_word in model.wv.index_to_key:
        # 跳过目标词本身
        if vocab_word == word:
            continue

        # 获取当前词的向量
        current_vec = model.wv[vocab_word]

        # 计算余弦相似度
        sim = cosine_similarity(target_vec, current_vec)

        # 存储
        similarities.append((vocab_word, sim))

    # 按相似度降序排序
    similarities.sort(key=lambda x: x[1], reverse=True)

    # 返回前topn个
    return similarities[:topn]

# 使用
result = most_similar('ship', model, topn=5)
print(result)
# [('boat', 0.9945),
#  ('head', 0.9881),
#  ('boiling', 0.9821),
#  ...]
```

四、GloVe模型

1. GloVe vs Word2Vec

Word2Vec：

> 基于上下文窗口
> 局部信息
>
> 例如：在"The cat sat"中
> 只看到cat和sat的局部共现

GloVe：

> 基于全局共现矩阵
> 全局统计信息
>
> 统计整个语料库中
> cat和sat共现了多少次

---

2. 加载预训练GloVe模型

```
import gensim.downloader as api

# 下载预训练的GloVe模型（Twitter数据）
# 这会下载大约1GB的文件
glove_model = api.load('glove-twitter-25')

# 使用方法和Word2Vec相同
similar_words = glove_model.most_similar('happy', topn=5)
print(similar_words)
# [('glad', 0.89),
#  ('pleased', 0.87),
#  ('excited', 0.85),
#  ...]
```

预训练模型的优点：

1. 无需自己训练
   → 节省时间和计算资源

2. 训练数据量大
   → 20亿Twitter推文
   → 覆盖更多词汇

3. 质量高
   → 专业团队训练
   → 参数优化好

五、文本分类应用

问题：如何用词向量分类句子?

挑战：

有词向量：
```
cat   = [0.2, 0.8, 0.1]
dog   = [0.3, 0.7, 0.2]
sat   = [0.5, 0.5, 0.5]

但句子怎么表示?
"The cat sat" = ?
```

解决方案：Pooling（池化）

Mean Pooling（平均池化）

方法：对所有词向量求平均

```
def mean_pooling(sentence, model):
    """
    计算句子的平均向量

    参数:
    sentence: 句子（词列表）
    model: 词向量模型

    返回:
    句子向量
    """
    # 存储所有词的向量
    vectors = []

    for word in sentence:
        # 如果词在模型中，获取其向量
        if word in model:
            vectors.append(model[word])

    # 如果没有词在模型中，返回零向量
    if len(vectors) == 0:
        return np.zeros(model.vector_size)

    # 计算平均值
    sentence_vec = np.mean(vectors, axis=0)

    return sentence_vec

# 例子
sentence = ['the', 'cat', 'sat']
sentence_vec = mean_pooling(sentence, glove_model)
print(sentence_vec.shape)  # (25,) 如果用25维GloVe
```

手工计算示例：

```
假设3维向量:
the = [0.1, 0.2, 0.3]
cat = [0.4, 0.5, 0.6]
sat = [0.7, 0.8, 0.9]
```

```
mean = (the + cat + sat) / 3
     = ([0.1, 0.2, 0.3] + [0.4, 0.5, 0.6] + [0.7, 0.8, 0.9]) / 3
     = [1.2, 1.5, 1.8] / 3
     = [0.4, 0.5, 0.6]
```

Max Pooling（最大池化）

```
def max_pooling(sentence, model):
    """最大池化：取每个维度的最大值"""
    vectors = []
    for word in sentence:
        if word in model:
            vectors.append(model[word])

    if len(vectors) == 0:
        return np.zeros(model.vector_size)

    sentence_vec = np.max(vectors, axis=0)
    return sentence_vec
```

手工计算示例：

```
the = [0.1, 0.2, 0.3]
cat = [0.4, 0.5, 0.6]
sat = [0.7, 0.8, 0.9]

max = [max(0.1, 0.4, 0.7),
       max(0.2, 0.5, 0.8),
       max(0.3, 0.6, 0.9)]
    = [0.7, 0.8, 0.9]
```

情感分类完整流程

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# 步骤1：加载数据
df = pd.read_csv('Tweets_short.csv')

# 步骤2：预处理
def preprocess_tweet(text):
    """简单的预处理"""
    # 转小写
    text = text.lower()
    # 分词
    tokens = text.split()
    return tokens

df['tokens'] = df['text'].apply(preprocess_tweet)

# 步骤3：提取句子向量
df['vector'] = df['tokens'].apply(
    lambda tokens: mean_pooling(tokens, glove_model)
)

# 步骤4：准备训练数据
X = np.vstack(df['vector'].values)  # 特征矩阵
y = df['sentiment'].values          # 标签

# 步骤5：划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 步骤6：训练分类器
classifier = SVC(kernel='linear')
classifier.fit(X_train, y_train)

# 步骤7：预测
y_pred = classifier.predict(X_test)

# 步骤8：评估
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2%}')

# 打印详细报告
print(classification_report(y_test, y_pred))
```

预处理Twitter特征

问题：Twitter文本有特殊元素

```
"@user I love #python! Check this out: http://..."
```

GloVe模型的预处理：

```
@user → <USER>
#python → <HASHTAG>
http://... → <URL>
```

实现：

```
import re

def preprocess_tweet_glove(text):
    """
    按照GloVe模型的预处理方式处理Twitter文本
    """
    # 替换URL
    text = re.sub(r'https?://\S+', '<URL>', text)

    # 替换@mentions
    text = re.sub(r'@\w+', '<USER>', text)

    # 替换#hashtags
    text = re.sub(r'#(\w+)', '<HASHTAG>', text)

    # 转小写
```

```
    text = text.lower()

    # 分词
    tokens = text.split()

    return tokens

# 例子
text = "@john I love #python! Check http://example.com"
print(preprocess_tweet_glove(text))
# ['<USER>', 'i', 'love', '<HASHTAG>', '!', 'check', '<URL>']
```

六、错误分析

混淆矩阵

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# 计算混淆矩阵
cm = confusion_matrix(y_test, y_pred,
                      labels=['positive', 'neutral', 'negative'])

# 可视化
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d',
            xticklabels=['positive', 'neutral', 'negative'],
            yticklabels=['positive', 'neutral', 'negative'])
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
plt.show()
```

分析错误样本

```
# 找出错误分类的样本
df_test = df.loc[X_test.index]
df_test['predicted'] = y_pred
df_test['correct'] = (y_test == y_pred)

# 错误样本
errors = df_test[df_test['correct'] == False]

# 分析：negative被预测为positive
neg_as_pos = errors[(errors['sentiment'] == 'negative') &
                    (errors['predicted'] == 'positive')]

print(f"Negative预测为Positive的样本数：{len(neg_as_pos)}")
print("\n示例：")
for idx, row in neg_as_pos.head(5).iterrows():
    print(f"文本：{row['text']}")
    print(f"真实：{row['sentiment']}, 预测：{row['predicted']}\n")
```

七、纸笔考试重点

必须掌握

1. ✅ 词嵌入的定义

```
定义：将词映射到低维稠密向量
目的：捕捉词的语义信息
```

2. ✅ 余弦相似度计算

```
公式：cos(θ) = (A·B) / (|A|×|B|)
```

会手工计算简单例子

3. ✅ 语义向量运算

```
king - man + woman ≈ queen

理解原理：向量包含语义特征
```

4. ✅ Word2Vec vs GloVe

```
Word2Vec：上下文窗口，局部信息
GloVe：共现矩阵，全局统计
```

5. ✅ Pooling方法

```
Mean Pooling：平均值
Max Pooling：最大值
Min Pooling：最小值

目的：将词向量转换为句子向量
```

可能的考题

题型1：余弦相似度计算

题目：计算以下两个向量的余弦相似度

```
v1 = [3, 4]
v2 = [6, 8]
```

▶ **答案**

题型2：Mean Pooling计算

题目：给定词向量，计算句子向量（mean pooling）

```
句子："cat sat"
cat = [1, 2, 3]
sat = [4, 5, 6]
```

▶ **答案**

题型3：理解语义向算

题目：解释为什么 `king - man + woman ≈ queen`

▶ **答案**

题型4：选择pooling方法

题目：以下场景应该使用哪种pooling?

```
场景1：识别句子中是否包含强烈情感词
场景2：计算句子的整体情感倾向
```

▶ **答案**

题型5：Word2Vec参数理解

题目：解释Word2Vec参数的含义

```
Word2Vec(vector_size=100, window=5, min_count=2)
```

▶ 答案

八、代码要点

1. Gensim基础

```
from gensim.models import Word2Vec

# 训练
model = Word2Vec(sentences, vector_size=100, window=5)

# 获取向量
vec = model.wv['word']

# 找相似词
similar = model.wv.most_similar('word', topn=10)

# 词类比
result = model.wv.most_similar(
    positive=['king', 'woman'],
    negative=['man']
)
```

---

2. 加载预训练模型

```
import gensim.downloader as api

# 列出可用模型
print(list(api.info()['models'].keys()))
```

```
# 加载模型
model = api.load('glove-twitter-25')
```

---

3. Pooling实现

```
# Mean pooling
sentence_vec = np.mean([model[w] for w in words], axis=0)

# Max pooling
sentence_vec = np.max([model[w] for w in words], axis=0)
```

---

4. SVM分类

```
from sklearn.svm import SVC

# 训练
clf = SVC(kernel='linear')
clf.fit(X_train, y_train)

# 预测
y_pred = clf.predict(X_test)

# 评估
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_pred)
```

九、记忆口诀

词嵌入

词向量低维稠密，
语义相似数值近，
分布假说是基础，
上下文定词意义。

向量运算

余弦相似看角度，
点积除以模长积，
king减man加woman，
语义运算queen出现。

Pooling

词向量变句向量，
平均最大最小三，
Mean适合看整体，
Max捕捉最强点。

十、常见错误

❌ 错误1：向量维度不匹配

```
# 错误
vec1 = model1.wv['word']  # 100维
vec2 = model2.wv['word']  # 300维
similarity = cosine_similarity(vec1, vec2)  # ❌ 维度不同

# 正确
# 确保使用同一个模型
vec1 = model.wv['word1']
```

```
vec2 = model.wv['word2']
similarity = cosine_similarity(vec1, vec2)  # ✅
```

❌ 错误2：忘记处理未登录词（OOV）

```
# 错误
sentence_vec = np.mean([model[w] for w in words])  # ❌
# 如果有词不在模型中，会报错

# 正确
vectors = []
for w in words:
    if w in model:  # 检查词是否在模型中
        vectors.append(model[w])

if len(vectors) > 0:
    sentence_vec = np.mean(vectors, axis=0)
else:
    sentence_vec = np.zeros(model.vector_size)  # ✅
```

❌ 错误3：使用错误的axis

```
# 错误
vectors = [v1, v2, v3]  # shape: (3, 100)
mean = np.mean(vectors, axis=1)  # ❌ 沿着第1维求平均
# 结果shape: (3,) 错误!

# 正确
mean = np.mean(vectors, axis=0)  # ✅ 沿着第0维求平均
# 结果shape: (100,) 正确!
```

祝你考试顺利！ 🎉

Lecture 8核心要点：

- 词嵌入：低维稠密向量表示词
- Word2Vec：基于上下文训练词向量
- GloVe：基于全局共现矩阵
- 余弦相似度：计算向量相似性
- 语义运算：king - man + woman ≈ queen
- Pooling：将词向量合并为句子向量（mean/max/min）
- 应用：用词嵌入做文本分类

记住：理解词嵌入的原理！会计算余弦相似度！会做pooling！