Lecture 6 超基础复习笔记 – 情感分析（Sentiment Analysis）

零、先理解问题（什么是情感分析？）

什么是情感分析?

定义：判断一段文本表达的情感是正面、负面还是中性

例子：

```
文本1: "I ❤ flying @VirginAmerica."
情感: 正面 (positive) ✅

文本2: "@SouthwestAir are you hiring for flight attendants right now?"
情感: 中性 (neutral) 😐

文本3: "@united I'm rebooked now, but the line was 300 people deep."
情感: 负面 (negative) ❌
```

为什么重要?

应用场景：

1. **商业分析** 分析客户对产品的评价 → 了解产品优缺点 → 改进产品和服务
2. **社交媒体监控** 分析用户对品牌的态度 → 及时发现公关危机 → 了解市场趋势
3. **政治分析** 分析选民对候选人的情感 → 预测选举结果

一、Lecture 6的两种方法

方法1：基于PMI（Pointwise Mutual Information）

核心思想：看词语和积极/消极词的共现关系

```
"great" 经常和积极词一起出现 → 积极
"terrible" 经常和消极词一起出现 → 消极
```

方法2：基于词典的可评分方法（Gradable Method）

核心思想：每个词都有一个情感分数

```
"love" → +3.5（很积极）
"like" → +1.2（较积极）
"hate" → -3.8（很消极）

句子情感分数 = 所有词的分数总和
```

二、PMI方法详解

1. PMI的定义

公式：

```
PMI(x, y) = log₂( P(x,y) / (P(x) × P(y)) )

其中：
- P(x,y) = x和y共现的概率
- P(x) = x出现的概率
- P(y) = y出现的概率
```

通俗理解：

```
PMI衡量：两个词一起出现的频率，是否高于"随机"情况

PMI > 0：比随机情况更常一起出现（正相关）
PMI = 0：和随机情况一样（不相关）
PMI < 0：比随机情况更少一起出现（负相关）
```

---

2. 为什么用PMI?

问题：如何判断一个词是积极还是消极?

传统方法：人工标注词典（费时费力）

PMI方法：自动从语料库学习

```
步骤：
1. 准备一些明确的积极词：love, great, fantastic
2. 准备一些明确的消极词：hate, bad, terrible
3. 对于任意词w，计算：
   - PMI(w, 积极词)
   - PMI(w, 消极词)
4. 比较两者，判断w的极性
```

例子：

```
词："excellent"

计算：
PMI("excellent", "love") = 2.5  ← 高
PMI("excellent", "hate") = -1.2 ← 低

结论："excellent" 更倾向于积极
```

---

3. 概率估计（用频率代替概率）

理论公式：

```
PMI(x, y) = log₂( P(x,y) / (P(x) × P(y)) )
```

实际计算：

```
P(x) ≈ C(x) / N
P(y) ≈ C(y) / N
P(x,y) ≈ C(x,y) / N

其中：
- C(x) = 包含词x的推文数量
- C(y) = 包含词y的推文数量
- C(x,y) = 同时包含x和y的推文数量
- N = 推文总数
```

代入：

```
PMI(x, y) = log₂( (C(x,y)/N) / ((C(x)/N) × (C(y)/N)) )
          = log₂( C(x,y) × N / (C(x) × C(y)) )
```

---

4. 手工计算示例

数据：

```
总推文数 N = 1000

"love" 出现在 100 条推文中
"JetBlue" 出现在 200 条推文中
"love" 和 "JetBlue" 共现在 50 条推文中
```

计算：

```
C(love) = 100
C(JetBlue) = 200
C(love, JetBlue) = 50
N = 1000

PMI(love, JetBlue) = log₂( 50 × 1000 / (100 × 200) )
```

```
                    = log₂( 50000 / 20000 )
                    = log₂( 2.5 )
                    ≈ 1.32
```

解释：

```
PMI = 1.32 > 0
→ "love" 和 "JetBlue" 正相关
→ "JetBlue" 有积极倾向
```

---

5. 代码实现

第1步：定义PMI函数

```
import numpy as np

def PMI(c_xy, c_x, c_y, N):
    """
    计算PMI(x, y)

    参数:
    c_xy: x和y共现的次数
    c_x: x出现的次数
    c_y: y出现的次数
    N: 总观测数（推文总数）

    返回:
    PMI值
    """
    # 计算概率
    p_xy = c_xy / N
    p_x = c_x / N
    p_y = c_y / N

    # 计算PMI
    pmi = np.log2( p_xy / (p_x * p_y) )

    return pmi
```

详细解释：

```
# 例子数据
c_xy = 50   # love和JetBlue共现50次
c_x = 100   # love出现100次
c_y = 200   # JetBlue出现200次
N = 1000    # 总共1000条推文

# 计算概率
p_xy = 50 / 1000 = 0.05      # 共现概率
p_x = 100 / 1000 = 0.1       # love概率
p_y = 200 / 1000 = 0.2       # JetBlue概率

# 计算PMI
pmi = np.log2( 0.05 / (0.1 * 0.2) )
    = np.log2( 0.05 / 0.02 )
    = np.log2( 2.5 )
    = 1.32
```

第2步：统计词频

```
from collections import Counter, defaultdict

# 出现次数统计
occ_counts = Counter()

# 共现次数统计（两层字典）
cooc_counts = defaultdict(Counter)

# 积极词列表
pos_words = ["love", "great", "like"]

# 消极词列表
neg_words = ["hate", "bad", "annoy"]

# 目标公司列表
companies = ["@virginamerica", "@united", "@southwestair"]

# 情感词 = 积极词 + 消极词
sentiment_words = pos_words + neg_words
```

数据结构解释：

```
# occ_counts：单个词的出现次数
occ_counts = Counter()
# 例如：
# occ_counts['love'] = 100
# occ_counts['@jetblue'] = 200

# cooc_counts：共现次数（两层字典）
cooc_counts = defaultdict(Counter)
# 第1层：公司名
# 第2层：情感词 → 共现次数
# 例如：
# cooc_counts['@jetblue']['love'] = 50
# cooc_counts['@jetblue']['hate'] = 10
```

第3步：统计数据

```
import pandas as pd

# 读取CSV文件
df = pd.read_csv("Tweets_short.csv", index_col=0)

# 转小写
df['text'] = df['text'].str.lower()

# 总推文数
N = len(df)

# 遍历每条推文
for sentiment, tweet in df.itertuples(index=False):
    # 分词（去重）
    words = set(tweet.strip().split())

    # 统计所有词的出现次数
    for word in words:
        occ_counts[word] += 1

    # 只统计公司和情感词的共现
    for word in words:
        if word in companies:
```

```
            for word2 in words:
                if word2 in sentiment_words:
                    cooc_counts[word][word2] += 1
```

详细解释：

为什么用 `set()`？

```
tweet = "I love love love @jetblue"
words_list = tweet.split()  # ['I', 'love', 'love', 'love', '@jetblue'
words_set = set(words_list)  # {'I', 'love', '@jetblue'}

# 用set去重的原因：
# 我们只关心"love"和"@jetblue"是否在同一条推文中
# 不关心"love"出现了几次
```

统计逻辑：

```
# 假设推文："I love @jetblue it's great"
words = {'i', 'love', '@jetblue', "it's", 'great'}

# 第1个循环：统计所有词的出现
for word in words:
    occ_counts[word] += 1
# 结果：
# occ_counts['i'] += 1
# occ_counts['love'] += 1
# occ_counts['@jetblue'] += 1
# occ_counts["it's"] += 1
# occ_counts['great'] += 1

# 第2个循环：统计共现
for word in words:
    if word in companies:  # 只处理公司名
        # word = '@jetblue'
        for word2 in words:
            if word2 in sentiment_words:  # 只处理情感词
                # word2 = 'love' 或 'great'
                cooc_counts[word][word2] += 1

# 结果：
```

```
# cooc_counts['@jetblue']['love'] += 1
# cooc_counts['@jetblue']['great'] += 1
```

第4步：计算PMI并判断情感

```
# 对每个公司
for company in companies:
    company_count = occ_counts[company]

    # 存储积极PMI和消极PMI
    posPMIs = []
    negPMIs = []

    # 计算和积极词的PMI
    for pos in pos_words:
        if pos in cooc_counts[company]:  # 检查是否共现
            pmi = PMI(cooc_counts[company][pos],
                      company_count,
                      occ_counts[pos],
                      N)
            posPMIs.append(pmi)

    # 计算和消极词的PMI
    for neg in neg_words:
        if neg in cooc_counts[company]:
            pmi = PMI(cooc_counts[company][neg],
                      company_count,
                      occ_counts[neg],
                      N)
            negPMIs.append(pmi)

    # 计算平均PMI
    avg_pos = mean(posPMIs)
    avg_neg = mean(negPMIs)

    print(f"{company}: {avg_pos:.2f} (pos), {avg_neg:.2f} (neg)")
```

执行示例：

```
# 假设数据:
company = '@jetblue'
occ_counts['@jetblue'] = 200

pos_words = ['love', 'great', 'like']
occ_counts['love'] = 100
occ_counts['great'] = 80
occ_counts['like'] = 150

cooc_counts['@jetblue']['love'] = 50
cooc_counts['@jetblue']['great'] = 30
cooc_counts['@jetblue']['like'] = 60

N = 1000

# 计算PMI
PMI_love = PMI(50, 200, 100, 1000)
         = log₂(50 × 1000 / (200 × 100))
         = log₂(2.5)
         = 1.32

PMI_great = PMI(30, 200, 80, 1000)
          = log₂(30 × 1000 / (200 × 80))
          = log₂(1.875)
          = 0.91

PMI_like = PMI(60, 200, 150, 1000)
         = log₂(60 × 1000 / (200 × 150))
         = log₂(2.0)
         = 1.0

# 平均PMI
avg_pos = (1.32 + 0.91 + 1.0) / 3 = 1.08

# 同样计算neg_words的PMI...
avg_neg = ...

# 输出
print("@jetblue: 1.08 (pos), -0.5 (neg)")
# 结论: @jetblue整体偏向积极（1.08 > -0.5）
```

---

6. PMI的意义

PMI值的解释：

```
PMI > 0：正相关
  → 两个词一起出现的频率，高于随机情况
  → 有某种联系

PMI = 0：不相关
  → 两个词一起出现的频率，等于随机情况
  → 没有特殊联系

PMI < 0：负相关
  → 两个词一起出现的频率，低于随机情况
  → 互相排斥
```

在情感分析中：

```
PMI(公司, 积极词) 高 → 公司评价积极
PMI(公司, 消极词) 高 → 公司评价消极

比较两者的平均值，判断整体情感
```

三、可评分方法（Gradable Method）

1. 核心思想

方法：给每个词分配一个情感分数

```
范围：-4（极度消极）到 +4（极度积极）

例子：
"excellent" → +3.5
"good"      → +2.0
"ok"        → +0.5
"bad"       → -2.0
"terrible"  → -3.5
```

句子情感：

```
句子情感分数 = Σ 每个词的情感分数

例如：
"The food is excellent"
= 0 + 0 + 0 + 3.5   （只有"excellent"有分数）
= 3.5（积极）

"The service is bad"
= 0 + 0 + 0 + (-2.0)
= -2.0（消极）
```

---

2. 情感词典（Valence Lexicon）

来源：人工标注

```
方法：
1. 让10个人对每个词打分（-4到+4）
2. 取平均值作为该词的情感分数
```

文件格式： `valence_lexicon_small.tsv`

```
word        Valence
love        3.06
good        2.52
great       3.32
hate        -2.79
bad         -2.90
terrible    -3.42
```

读取代码：

```
import pandas as pd

# 读取TSV文件
valence = pd.read_csv("valence_lexicon_small.tsv",
                      sep='\t',
```

```
                        index_col=0)['Valence'].to_dict()

# 结果是字典
# valence = {
#     'love': 3.06,
#     'good': 2.52,
#     'great': 3.32,
#     'hate': -2.79,
#     ...
# }
```

---

3. 基础算法

```
def classify_sentiment_basic(tweet, valence):
    """
    基础情感分类

    参数:
    tweet: 推文文本
    valence: 情感词典 (字典)

    返回:
    'positive', 'neutral', 或 'negative'
    """
    # 分词
    words = tweet.strip().split()

    # 累加情感分数
    tweet_valence = 0
    for w in words:
        if w in valence:
            tweet_valence += valence[w]

    # 根据阈值分类
    threshold = 0.05

    if tweet_valence > threshold:
        return 'positive'
    elif tweet_valence < -threshold:
        return 'negative'
```

```
    else:
        return 'neutral'
```

执行示例：

```
tweet = "I love this airline it's great"
valence = {
    'love': 3.06,
    'great': 3.32,
    # ... 其他词
}

# 分词
words = ['i', 'love', 'this', 'airline', "it's", 'great']

# 累加
tweet_valence = 0
# 'i' 不在词典 → 跳过
# 'love' 在词典 → tweet_valence += 3.06
tweet_valence = 3.06
# 'this' 不在词典 → 跳过
# 'airline' 不在词典 → 跳过
# "it's" 不在词典 → 跳过
# 'great' 在词典 → tweet_valence += 3.32
tweet_valence = 3.06 + 3.32 = 6.38

# 分类
threshold = 0.05
6.38 > 0.05  # True
→ 'positive'
```

---

4. 阈值（Threshold）的作用

问题：情感分数接近0时如何分类?

```
例子:
tweet_valence = 0.01  ← 几乎是中性，但稍微积极
应该分类为 'positive' 还是 'neutral'?
```

解决方案：设置阈值

```
threshold = 0.05

if valence > +0.05:
    → positive
elif valence < -0.05:
    → negative
else:  # -0.05 <= valence <= +0.05
    → neutral
```

图示：

```
        -0.05        0        +0.05
  --------|----------|----------|--------
  negative  neutral    neutral   positive
  ←————————|←————————→|————————→
```

阈值的影响：

```
threshold = 0.01（小阈值）
→ neutral范围窄
→ 更多推文被分类为positive/negative
→ 分类更"敏感"

threshold = 1.0（大阈值）
→ neutral范围宽
→ 更多推文被分类为neutral
→ 分类更"保守"
```

---

5. 高级特性

特性1：否定词处理

问题：

```
"I love this"        → positive ✅
```

```
"I don't love this"  → 也是positive? ❌ (错了!)
```

解决方案：识别否定词，反转情感

```
negation_words = ['not', "don't", "didn't", "never", 'no']

# 修改算法
for w in words:
    if w in valence:
        tweet_valence += valence[w]
    if w in negation_words:
        tweet_valence *= -0.8  # 反转并减弱
```

例子：

```
tweet = "I don't love this"
words = ['i', "don't", 'love', 'this']

tweet_valence = 0
# 'i' → 跳过
# "don't" → 否定词
tweet_valence *= -0.8  # 0 * -0.8 = 0
# 'love' → valence['love'] = 3.06
tweet_valence += 3.06  # 0 + 3.06 = 3.06
# 但前面有否定词，最终应该是负的

# 更好的实现:
tweet_valence = 0
negation_flag = False
for w in words:
    if w in negation_words:
        negation_flag = True
    if w in valence:
        if negation_flag:
            tweet_valence += -valence[w] * 0.8
        else:
            tweet_valence += valence[w]

# 结果:
# "don't" → negation_flag = True
# 'love' → tweet_valence += -3.06 * 0.8 = -2.45
# 最终: tweet_valence = -2.45 (负面) ✅
```

特性2：加强词（Strengthening Words）

作用：增强情感强度

```
"good"      → +2.0
"very good" → +2.0 + 0.5 = +2.5（更积极）
```

加强词列表：

```
strengthen_words = {
    'very': 0.5,
    'really': 0.5,
    'extremely': 0.8,
    'absolutely': 0.8
}
```

算法：

```
for w in words:
    if w in valence:
        tweet_valence += valence[w]
    if w in strengthen_words:
        tweet_valence += strengthen_words[w]
```

特性3：减弱词（Weakening Words）

作用：减弱情感强度

```
"good"          → +2.0
"somewhat good" → +2.0 - 0.3 = +1.7（较不积极）
```

减弱词列表：

```
weaken_words = {
    'somewhat': -0.3,
```

```
    'slightly': -0.3,
    'barely': -0.5
}
```

特性4：感叹号

作用：增强情感

```
"Great"    → +3.32
"Great!"   → +3.32 + 0.1 = +3.42
"Great!!!" → +3.32 + 0.3 = +3.62
```

实现：

```
exclamation_words = {
    '!': 0.1,
    '!!': 0.2,
    '!!!': 0.3
}
```

---

6. 完整算法

```
def classify_sentiment_advanced(tweet, valence,
                                negation_words,
                                strengthen_words,
                                weaken_words,
                                exclamation_words):
    """
    高级情感分类（考虑否定、加强、减弱等）
    """
    words = tweet.strip().split()
    tweet_valence = 0

    for w in words:
        # 基础情感分数
        if w in valence:
```

```
            tweet_valence += valence[w]

        # 否定词
        if w in negation_words:
            tweet_valence *= -0.8

        # 感叹号
        if w in exclamation_words:
            tweet_valence += exclamation_words[w]

        # 加强词
        if w in strengthen_words:
            tweet_valence += strengthen_words[w]

        # 减弱词
        if w in weaken_words:
            tweet_valence += weaken_words[w]

    # 分类
    threshold = 0.05
    if tweet_valence > threshold:
        return 'positive'
    elif tweet_valence < -threshold:
        return 'negative'
    else:
        return 'neutral'
```

四、评估指标

1. 混淆矩阵（Confusion Matrix）

定义：对比真实标签和预测标签

```
              预测
            pos   neu   neg
真   pos   [  50    10     5 ]
实   neu   [  10   100    15 ]
     neg   [   3    12    60 ]
```

含义：

```
第1行：真实是positive的65个样本
  - 50个预测正确（pos → pos）
  - 10个预测错误（pos → neu）
  - 5个预测错误（pos → neg）

第2行：真实是neutral的125个样本
  - 10个预测为pos
  - 100个预测正确
  - 15个预测为neg

第3行：真实是negative的75个样本
  - 3个预测为pos
  - 12个预测为neu
  - 60个预测正确
```

---

2. 准确率（Accuracy）

公式：

```
准确率 = 正确预测的数量 / 总数量
       =（对角线元素之和）/（所有元素之和）
```

计算示例：

```
混淆矩阵：
            pos  neu  neg
       pos [50   10    5]
       neu [10  100   15]
       neg [ 3   12   60]

正确预测 = 50 + 100 + 60 = 210
总数 = 65 + 125 + 75 = 265

准确率 = 210 / 265 = 0.792 = 79.2%
```

---

3. 代码实现

```
import numpy as np

# 初始化混淆矩阵（3×3）
cm = np.zeros((3, 3))

# 情感标签到数字的映射
def s2id(sentiment):
    if sentiment == "positive":
        return 0
    elif sentiment == "neutral":
        return 1
    elif sentiment == "negative":
        return 2

# 遍历每条推文
for true_sentiment, tweet in df.itertuples(index=False):
    # 预测情感
    pred_sentiment = classify_sentiment(tweet)

    # 更新混淆矩阵
    cm[s2id(true_sentiment)][s2id(pred_sentiment)] += 1

# 计算准确率
accuracy = np.trace(cm) / np.sum(cm)
print(f"Accuracy: {accuracy:.2%}")

# 打印混淆矩阵
print(cm)
```

详细解释：

```
# 假设预测结果：
true_sentiment = 'positive'
pred_sentiment = 'neutral'

# 更新混淆矩阵
true_id = s2id('positive') = 0
pred_id = s2id('neutral') = 1

cm[0][1] += 1
# 表示：真实是positive，预测为neutral
```

```
# 混淆矩阵的含义：
# cm[i][j] = 真实标签为i，预测标签为j的数量

# np.trace(cm)：对角线元素之和
# 例如：cm[0][0] + cm[1][1] + cm[2][2]
# 这些是预测正确的样本

# np.sum(cm)：所有元素之和（总样本数）
```

五、完整执行流程

PMI方法流程

```
步骤1：准备数据
  ├─ 读取tweets CSV文件
  ├─ 转小写
  └─ 定义积极词、消极词、目标公司

步骤2：统计词频
  ├─ 统计每个词的出现次数（occ_counts）
  └─ 统计公司和情感词的共现次数（cooc_counts）

步骤3：计算PMI
  ├─ 对每个公司
  │   ├─ 计算和积极词的PMI
  │   └─ 计算和消极词的PMI
  └─ 求平均值

步骤4：判断情感
  └─ 比较avg_pos和avg_neg
```

可评分方法流程

```
步骤1：准备数据
  ├─ 读取tweets CSV文件
  ├─ 读取情感词典
```

```
   └─ 读取否定词、加强词等列表

步骤2：对每条推文
   ├─ 分词
   ├─ 累加情感分数
   ├─ 处理否定、加强、减弱等
   └─ 根据阈值分类

步骤3：评估
   ├─ 构建混淆矩阵
   ├─ 计算准确率
   └─ 可视化结果
```

六、两种方法的对比

PMI方法

优点：

```
1. 无需人工标注词典
2. 能发现意想不到的关联
3. 适合分析整体情感（公司、产品等）
```

缺点：

```
1. 需要大量数据
2. 只能判断整体，不能逐句分类
3. 依赖种子词（积极词和消极词列表）
```

可评分方法

优点：

```
1. 可以对每条推文分类
2. 可以处理复杂语言现象（否定、加强等）
```

```
3. 结果可解释（可以看每个词的贡献）
```

缺点：

```
1. 需要人工标注的情感词典
2. 词典覆盖不全的词无法处理
3. 简单相加可能不准确
```

七、纸笔考试重点

必须掌握

1. ✅ PMI的定义和计算

```
公式：
PMI(x,y) = log₂( P(x,y) / (P(x) × P(y)) )
         = log₂( (C(x,y) × N) / (C(x) × C(y)) )

能手工计算给定数据的PMI值
```

2. ✅ PMI值的意义

```
PMI > 0：正相关（一起出现比随机多）
PMI = 0：不相关
PMI < 0：负相关（一起出现比随机少）
```

3. ✅ 可评分方法的原理

```
句子情感 = Σ 词的情感分数
根据阈值分类为positive/neutral/negative
```

4. ✅ 混淆矩阵

```
能读懂混淆矩阵
能计算准确率 = 对角线之和 / 总和
```

5. ✅ 特殊词的处理

```
否定词：反转情感
加强词：增强情感
减弱词：减弱情感
感叹号：增强情感
```

可能的考题

题型1：计算PMI

题目：给定以下数据，计算PMI(excellent, restaurant)

```
总推文数：N = 10,000
"excellent" 出现：500次
"restaurant" 出现：1,000次
两者共现：150次
```

▶ 答案

题型2：PMI的意义

题目：解释以下PMI值的含义

```
PMI(love, @JetBlue) = 2.5
PMI(hate, @JetBlue) = -0.5
```

▶ 答案

题型3：可评分方法分类

题目：使用可评分方法对以下推文分类

```
推文："I really love this airline!"
情感词典：
  love → +3.0
  really → +0.5（加强词）
  ! → +0.1（感叹号）

阈值：threshold = 0.5
```

▶ 答案

题型4：否定词处理

题目：考虑否定词后，以下推文应该分类为什么？

```
推文："I don't hate it"
情感词典：
  hate → -3.0
否定词列表：["don't"]
否定规则：遇到否定词，将后续情感分数乘以-0.8

阈值：threshold = 0.5
```

▶ 答案

题型5：混淆矩阵和准确率

题目：给定混淆矩阵，计算准确率

```
            预测
          pos  neu  neg
真  pos [ 40   10    5]
```

```
实  neu [  5  80  10]
    neg [  2   8  50]
```

▶ 答案

八、Python库和函数

1. Collections

```
from collections import Counter, defaultdict

# Counter：计数字典
counter = Counter(['a', 'b', 'a', 'c', 'b', 'a'])
# Counter({'a': 3, 'b': 2, 'c': 1})

counter.most_common(2)
# [('a', 3), ('b', 2)]  ← 返回最常见的2个

# defaultdict：默认值字典
dd = defaultdict(Counter)
dd['key1']['subkey'] += 1  # 自动创建Counter
```

---

2. Pandas

```
import pandas as pd

# 读取CSV
df = pd.read_csv("file.csv", index_col=0)

# 读取TSV
df = pd.read_csv("file.tsv", sep='\t', index_col=0)

# 转小写
df['text'] = df['text'].str.lower()

# 遍历
```

```
for sentiment, text in df.itertuples(index=False):
    print(sentiment, text)

# 转字典
valence = df['Valence'].to_dict()
```

---

3. NumPy

```
import numpy as np

# 对数
np.log2(8)  # 3.0

# 矩阵
matrix = np.zeros((3, 3))  # 3×3零矩阵

# 对角线之和
np.trace(matrix)

# 所有元素之和
np.sum(matrix)

# 平均值
np.mean([1, 2, 3])  # 2.0
```

九、记忆口诀

PMI方法

```
共现频率看关联，
PMI计算要记全，
对数公式莫搞混，
正负零值分得清。
```

可评分方法

```
词典分数来判断，
阈值设置很关键，
否定反转要注意，
加强减弱别忘记。
```

混淆矩阵

```
对角线上是正确，
其他格子是错误，
总和一除得准确，
三类分开算精度。
```

十、常见错误

❌ 错误1：PMI计算公式错误

```
# 错误
PMI = log(c_xy / (c_x * c_y))  ❌
# 忘记乘N，忘记log的底数

# 正确
PMI = np.log2((c_xy * N) / (c_x * c_y))  ✅
```

❌ 错误2：共现统计不去重

```
# 错误
tweet = "I love love love @jetblue"
words = tweet.split()  # ['I', 'love', 'love', 'love', '@jetblue']
# love和@jetblue共现3次? ❌

# 正确
words = set(tweet.split())  # {'I', 'love', '@jetblue'}
# love和@jetblue共现1次✅
# 因为我们只关心是否在同一条推文中，不关心出现几次
```

❌ 错误3：阈值设置不合理

```
# 不合理
threshold = 0   # 太小，没有neutral
# 所有接近0的都被分为positive或negative

threshold = 10   # 太大，全是neutral
# 只有极端情感才被分类

# 合理
threshold = 0.05   # 适中
# 给neutral留有空间，又不至于太宽
```

❌ 错误4：混淆矩阵行列搞反

```
# 错误
cm[pred_id][true_id] += 1  ❌
# 预测在行，真实在列（反了！）

# 正确
cm[true_id][pred_id] += 1  ✅
# 真实在行，预测在列

# 记忆：True Real（真实在行）
```

十一、扩展应用

1. 多分类情感分析

```
除了positive/neutral/negative，还可以：
- 5分类：very negative, negative, neutral, positive, very positive
- 情绪分类：happy, sad, angry, fearful, surprised
```

2. 方面情感分析（Aspect-Based Sentiment）

不只判断整体情感，还要判断对具体方面的情感

例如餐厅评论：
"The food is excellent but the service is terrible"
→ food: positive
→ service: negative

---

3. 实时情感监控

应用：
1. 监控品牌声誉
2. 跟踪产品发布反应
3. 预警公关危机

祝你考试顺利！ 🎉

Lecture 6核心要点： – PMI公式：`log₂(C(x,y)×N / (C(x)×C(y)))` – PMI > 0正相关，< 0负相关 – 可评分方法：累加词的情感分数 – 阈值分类：positive > threshold, negative < –threshold – 否定词反转情感 – 混淆矩阵评估：准确率 = 对角线和/总和

记住：理解PMI的数学含义，会手工计算！理解否定、加强等语言现象的处理方法！