Python文本挖掘通用流程 (Lecture 12)

0. 启动Jupyter

Terminal中:

```
jupyter notebook
```

1. 必要库导入

```
# 基础数据处理
import pandas as pd
import numpy as np

# 文本处理和分析
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from bs4 import BeautifulSoup
import re

# 可视化
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# 机器学习相关
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

2. 文本预处理

2.1 文本清洗

```
# HTML标签清理
def clean_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# URL清理
def remove_urls(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
```

```
    return url_pattern.sub(r'', text)

# 特殊字符清理
def clean_special_chars(text):
    # 只保留字母和空格
    return re.sub(r'[^a-zA-Z\s]', '', text)

# 多余空白清理
def clean_whitespace(text):
    return ' '.join(text.split())

# 大小写统一
def normalize_case(text):
    return text.lower()

# 应用所有清理步骤
def clean_text(text):
    text = clean_html(text)
    text = remove_urls(text)
    text = clean_special_chars(text)
    text = clean_whitespace(text)
    text = normalize_case(text)
    return text
```

2.2 文本分词

```
# 下载必要的NLTK资源
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# 句子分词
def sentence_tokenize(text):
    return sent_tokenize(text)

# 词语分词
def word_tokenize_text(text):
    return word_tokenize(text)

# 停用词过滤
def remove_stopwords(tokens):
    stop_words = set(stopwords.words('english'))
    return [w for w in tokens if w not in stop_words]

# 词干提取
def stem_words(tokens):
    stemmer = PorterStemmer()
    return [stemmer.stem(word) for word in tokens]
```

```
# 词形还原
def lemmatize_words(tokens):
    lemmatizer = WordNetLemmatizer()
    return [lemmatizer.lemmatize(word) for word in tokens]
```

2.3 特征提取

```
# 词袋模型 (BoW)
def create_bow(texts):
    vectorizer = CountVectorizer(min_df=2, max_df=0.95)
    bow_matrix = vectorizer.fit_transform(texts)
    return vectorizer, bow_matrix

# TF-IDF向量化
def create_tfidf(texts):
    tfidf = TfidfVectorizer(min_df=2, max_df=0.95)
    tfidf_matrix = tfidf.fit_transform(texts)
    return tfidf, tfidf_matrix

# N-gram特征
def create_ngram_features(texts, ngram_range=(1,2)):
    ngram_vectorizer = CountVectorizer(ngram_range=ngram_range)
    ngram_matrix = ngram_vectorizer.fit_transform(texts)
    return ngram_vectorizer, ngram_matrix
```

3. 文本分析

3.1 基本统计分析

```
def text_statistics(df, text_column):
    # 文本长度统计
    df['text_length'] = df[text_column].str.len()

    # 词数统计
    df['word_count'] = df[text_column].str.split().str.len()

    # 句子数统计
    df['sentence_count'] = df[text_column].apply(lambda x: len(sent_tokenize(x)))

    # 平均词长统计
    df['avg_word_length'] = df[text_column].apply(lambda x:
        np.mean([len(w) for w in str(x).split()]))

    return df
```

```
# 查看基本统计量
print(df[['text_length', 'word_count', 'sentence_count', 'avg_word_length']].describe())
```

3.2 词频分析

```
def get_word_frequencies(texts):
    # 创建词频向量器
    vectorizer = CountVectorizer()
    word_freq_matrix = vectorizer.fit_transform(texts)

    # 获取词频统计
    word_freq = pd.DataFrame(word_freq_matrix.sum(axis=0).T,
                             index=vectorizer.get_feature_names_out(),
                             columns=['frequency'])

    return word_freq.sort_values('frequency', ascending=False)

# 绘制词频分布
def plot_top_words(word_freq, top_n=20):
    plt.figure(figsize=(12, 6))
    sns.barplot(data=word_freq.head(top_n),
                y=word_freq.head(top_n).index,
                x='frequency')
    plt.title(f'Top {top_n} Most Frequent Words')
    plt.xlabel('Frequency')
    plt.ylabel('Words')
    plt.show()
```

3.3 文本相似度分析

```
from sklearn.metrics.pairwise import cosine_similarity

def calculate_text_similarity(tfidf_matrix):
    # 计算文档间的余弦相似度
    similarity_matrix = cosine_similarity(tfidf_matrix)

    # 创建相似度DataFrame
    similarity_df = pd.DataFrame(similarity_matrix)

    return similarity_df

# 可视化相似度矩阵
def plot_similarity_matrix(similarity_df):
    plt.figure(figsize=(10, 8))
    sns.heatmap(similarity_df, cmap='YlOrRd')
    plt.title('Document Similarity Matrix')
```

```
    plt.show()
```

4. 文本分类

4.1 数据准备

```
# 准备特征矩阵和标签
X = tfidf_matrix
y = df['label']  # 假设有标签列

# 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)
```

4.2 模型训练与评估

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression

# 训练朴素贝叶斯分类器
def train_naive_bayes(X_train, y_train):
    nb_classifier = MultinomialNB()
    nb_classifier.fit(X_train, y_train)
    return nb_classifier

# 训练逻辑回归分类器
def train_logistic_regression(X_train, y_train):
    lr_classifier = LogisticRegression()
    lr_classifier.fit(X_train, y_train)
    return lr_classifier

# 模型评估
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)

    # 打印评估指标
    print("Classification Report:")
    print(classification_report(y_test, y_pred))

    # 绘制混淆矩阵
    plt.figure(figsize=(8, 6))
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d')
    plt.title('Confusion Matrix')
    plt.show()
```

4.3 文本聚类

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

def cluster_texts(tfidf_matrix, n_clusters=5):
    # 应用K-means聚类
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    clusters = kmeans.fit_predict(tfidf_matrix)

    # 降维以便可视化
    pca = PCA(n_components=2)
    coords = pca.fit_transform(tfidf_matrix.toarray())

    # 可视化聚类结果
    plt.figure(figsize=(10, 8))
    scatter = plt.scatter(coords[:, 0], coords[:, 1], c=clusters)
    plt.colorbar(scatter)
    plt.title('Text Clusters Visualization')
    plt.show()

    return clusters
```

5. 主题建模

5.1 LDA主题模型

```
from sklearn.decomposition import LatentDirichletAllocation

def apply_lda(tfidf_matrix, n_topics=5, n_words=10):
    # 训练LDA模型
    lda = LatentDirichletAllocation(n_components=n_topics, random_state=42)
    lda_output = lda.fit_transform(tfidf_matrix)

    # 获取每个主题的关键词
    feature_names = tfidf.get_feature_names_out()
    for topic_idx, topic in enumerate(lda.components_):
        top_words = [feature_names[i] for i in topic.argsort()[:-n_words-1:-1]]
        print(f"Topic {topic_idx + 1}:", ", ".join(top_words))

    return lda, lda_output
```

5.2 主题可视化

```
def visualize_topics(lda_output):
    # 创建主题分布图
    plt.figure(figsize=(12, 6))
    topic_shares = lda_output.mean(axis=0)
    plt.bar(range(len(topic_shares)), topic_shares)
    plt.title('Topic Distribution in Corpus')
    plt.xlabel('Topic')
    plt.ylabel('Share')
    plt.show()
```

6. 情感分析

6.1 基于词典的情感分析

```
from textblob import TextBlob
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

def analyze_sentiment(texts):
    # TextBlob情感分析
    textblob_sentiments = [TextBlob(text).sentiment.polarity for text in texts]

    # VADER情感分析
    analyzer = SentimentIntensityAnalyzer()
    vader_sentiments = [analyzer.polarity_scores(text)['compound'] for text in texts]

    return pd.DataFrame({
        'text': texts,
        'textblob_sentiment': textblob_sentiments,
        'vader_sentiment': vader_sentiments
    })
```

6.2 情感可视化

```
def plot_sentiment_distribution(sentiment_df):
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.hist(sentiment_df['textblob_sentiment'], bins=50)
    plt.title('TextBlob Sentiment Distribution')

    plt.subplot(1, 2, 2)
    plt.hist(sentiment_df['vader_sentiment'], bins=50)
    plt.title('VADER Sentiment Distribution')

    plt.tight_layout()
    plt.show()
```