数据预处理与情感分类模块

如果清洗的是英文文本数据，并在此基础上进行自然语言处理（NLP）的分析，以下是一般的英文文本数据清洗步骤：

1. 去除特殊字符：删除数据中不需要的符号和字符，包括标点、符号和HTML标签等，以确保文本数据为纯文本。
2. 去除停用词：去除不必要的单词和停用词，例如“a”、“the”，这些词对文本分析没有实际的贡献。
3. 标准化单词：将文本数据中的词汇规范化，例如将不同的大小写、时态或形式的同一单词标准化为统一的形式。
4. 词干提取和词形还原：对文本数据中的词汇进行词干提取（stemming）和词形还原（lemmatization）操作，以减少文字内容中的冗余，并使其更加高效。
5. 实体识别：如果文本涉及具体实体或命名实体，需要进行实体识别操作。
6. 拼写检查：进行拼写检查操作，以识别和修正数据中存在的拼写错误。
7. 情感分析：运用NLP工具，进行情感分析，识别文本中的正面、负面或中性情感，并为其打分。

以上是一般的英文文本数据清洗步骤，具体的清洗过程取决于数据集本身的特点和实际需求。需要注意的是，文本分析是一个复杂的任务，并且只有在为文本建立正确的清洗步骤并清洗好数据后才能得到准确的分析结果。

首先去标点符号：

```
#去掉标点符号，以及机械压缩
def preprocess_word(word):
    # Remove punctuation
    word = word.strip('\'"?!,.():;')
    # Convert more than 2 letter repetitions to 2 letter
    # funnnnny --> funny
    word = re.sub(r'(.)\1+', r'\1\1', word)
    # Remove - & '
    word = re.sub(r'(-|\')', '', word)
    return word

#再次去掉标点符号
def gettext(x):
    import string
    punc = string.punctuation
    for ch in punc:
        txt = str(x).replace(ch,"")
    return txt
```

接着去停用词：

```
stop_words = []
with open('常用英文停用词(NLP处理英文必备)stopwords.txt','r',encoding='utf-8')as f:
    lines = f.readlines()
    for line in lines:
        stop_words.append(line.strip().replace("'","""))
```

再替换表情包，在数据中，会有很多表情包，有时候这些表情包容易影响文本的判断，只好去掉

```
#替换表情包
def handle_emojis(tweet):
    # Smile -- :), : ), :-), (:, ( :, (-:, :')
    tweet = re.sub(r'(:\s?\)|:-\)|\(\s?:|\(-:|:\'\))', ' ', tweet)
    # Laugh -- :D, : D, :-D, xD, x-D, XD, X-D
    tweet = re.sub(r'(:\s?D|:-D|x-?D|X-?D)', ' ', tweet)
    # Love -- <3, :*
    tweet = re.sub(r'(<3|:\*)', ' ', tweet)
    # Wink -- ;-), ;), ;-D, ;D, (;,  (-;
    tweet = re.sub(r'(;-?\)|;-?D|\(-?;)', ' ', tweet)
    # Sad -- :-(, : (, :(, ):, )-:
    tweet = re.sub(r'(:\s?\(|:-\(|\)\s?:|\)-:)', ' ', tweet)
    # Cry -- :,(, :'(, :"(
    tweet = re.sub(r'(:,\(|:\'\(|:"\()', ' ', tweet)
    # Angry -- >:(, >:-(, :'(
    tweet = re.sub(r'(>:\(|>:-\(|:\'\()', ' ', tweet)
    # Surprised -- :O, :o, :-O, :-o, :0, 8-0
    tweet = re.sub(r'(:\s?[oO]|:-[oO]|:0|8-0)', ' ', tweet)
    # Confused -- :/, :\, :-/, :-\
    tweet = re.sub(r'(:\\|:/|:-\\\\|:-/)', ' ', tweet)
    # Embarrassed -- :$, :-$
    tweet = re.sub(r'(:\$|:-\$)', ' ', tweet)
    # Other emoticons
    # This regex matching method is taken from Twitter NLP utils:
    # https://github.com/aritter/twitter_nlp/blob/master/python/emoticons.py
    emoticons = re.findall(r'(?::|;|=)(?:-)?(?:\)|\(|D|P)', tweet)
    for emoticon in emoticons:
        tweet = tweet.replace(emoticon, " ")
    return tweet
```

接着先进行标准化单词 再对词干提取和词形还原：

```
def clean_text(text):
    # Replaces URLs with the word URL
    text = re.sub(r'((www\.[\S]+)|(https?://[\S]+))', ' URL ', text)
    # Replace @username with the word USER_MENTION
    text = re.sub(r'@[\S]+', ' __USER_MENTION__ ', text)
    # Replace #hashtag with the word HASHTAG
    text = re.sub(r'#(\S+)', ' __HASHTAG__ ', text)
    # Remove RT (retweet)
    text = re.sub(r'\brt\b', ' ', text)
    # Replace 2+ dots with space
```

```
    text = re.sub(r'\.{2,}', ' ', text)
    # Strip space, " and ' from text
    text = text.strip(' "\'')
    # Handle emojis
    text = handle_emojis(text)
    # Replace multiple spaces with a single space
    text = re.sub(r'\s+', ' ', text)
    # Remove numbers
    text = re.sub(r'\d+', ' ', text)
    # Remove punctuation
    text = re.sub(r'[^A-Za-z0-9\s]+', ' ', text)
    # Lowercase and split into words
    words = text.lower().split()
    words = [w for w in words if w not in stop_words]
    words = [lemmatizer.lemmatize(w) for w in words]
    words = [preprocess_word(w) for w in words]
    if len(words) != 0:
        return ' '.join(words)
    else:
        return np.NAN
```

在做完上面的步骤之后，我们就可以进行下面的操作

情感分析：

NLTK（Natural Language Toolkit）是一个流行的Python自然语言处理库，提供了一系列用于文本分析和处理的工具和方法。

具体代码如下：

通过对它的分值进行计算，再进行打分

```
sid = SentimentIntensityAnalyzer()

def emotional_judgment(x):
    neg = x['neg']
    neu = x['neu']
    pos = x['pos']
    compound = x['compound']
    if compound == 0 and neg == 0 and pos == 0 and neu == 1:
        return 'neu'
    if compound > 0:
        if pos > neg:
            return 'pos'
        else:
            return 'neg'
    elif compound < 0:
        if pos < neg:
            return 'neg'
        else:
            return 'pos'
```

接着我们根据上面处理好的数据进行LDA主题建模：

LDA(Latent Dirichlet Allocation)是一种主题模型，通常用于文本分析中。在聚类分析中，使用LDA的目的是对数据进行主题建模，从而捕捉数据的隐含特征，提高聚类效果。LDA能够根据数据内在结构，自动发现“话题”，对于探索文本主题或按照主题进行文本分类分析具有较强的实用价值。

LDA具有以下意义：

挖掘数据内在结构：在聚类分析中，LDA可以通过挖掘样本数据背后的主题，识别隐藏于数据中的内在结构，从而能够更好地理解数据特征之间的相关性和相互关系。

降低数据维度：在聚类分析中，为了避免过度拟合数据、提高模型泛化能力，通常需要对数据进行降维处理。LDA可以提取数据中的主题特征，从而降低数据维数，减少数据处理的复杂度。

提高聚类精度：通过LDA建模分析，可以对文本数据进行更精细的划分，将数据划分为相似的主题类别。基于这种类别划分，可以建立更准确的聚类模型。LDA也可以基于主题相似性，自动进行聚类，从而提高聚类精度。

数据可视化：LDA减小了高维度的数据量，更容易进行可视化建模，可使聚类结果更有可解释性。同时，LDA也可以通过可视化技术，可视化各类主题和其相关的文档，便于分析者进行数据挖掘、数据分析甚至决策。

通过LDA方法进行聚类分析，能够在一定程度上缓解高维数据的复杂度，提高聚类的准确性，进而增强聚类分析的可理解性、可解释性，具有良好的可视化效果。因此，在聚类分析中，采用LDA方法对数据进行主题建模，可以帮助我们更好的理解数据的内在结构和相关性，获得更高效和更精确的聚类结果。

具体代码：

```
dictionary = corpora.Dictionary([text.split() for text in df['clearn_comment']])
corpus = [dictionary.doc2bow(text.split()) for text in df['clearn_comment']]


num_topics = input('请输入主题数:')

#LDA可视化模块
#构建lda主题参数
lda = gensim.models.ldamodel.LdaModel(corpus=corpus, id2word=dictionary,
num_topics=num_topics, random_state=111, iterations=400)
#读取lda对应的数据
data1 = pyLDAvis.gensim.prepare(lda, corpus, dictionary)
#把数据进行可视化处理
pyLDAvis.save_html(data1, './result2/lda.html')

# 主题判断模块
list3 = []
list2 = []
# 这里进行lda主题判断
for i in lda.get_document_topics(corpus)[:]:
    listj = []
    list1 = []
    for j in i:
        list1.append(j)
        listj.append(j[1])
    list3.append(list1)
    bz = listj.index(max(listj))
    list2.append(i[bz][0])
```

```
df['主题概率'] = list3
df['主题类型'] = list2

df.to_csv('./result2/new_data.csv', encoding='utf-8-sig', index=False)
```

接着我们把处理好的主题，与信息熵结合一下

信息熵是信息论中的一个概念，用于度量一个随机变量的不确定性或者混乱程度。它反映了一个事件发生时所提供的平均信息量。在文本数据中，信息熵可以用来衡量文本的多样性和信息量。

在信息论中，信息熵的计算公式如下：

H(X) = - Σ P(xi) * log2(P(xi))

其中，H(X)表示随机变量X的信息熵，P(xi)表示随机变量X取某个特定取值xi的概率，log2表示以2为底的对数。信息熵的单位通常是比特（bit），表示所需的信息量。

将信息熵应用于文本数据时，可以将文本看作是一个随机变量，每个单词或者字符都是该变量的一个可能取值。通过计算单词或者字符的出现概率，可以得到文本的信息熵。

在文本分析中，信息熵可以用来：

1. 衡量文本的多样性：信息熵越大，表示文本中的单词或者字符分布越均匀，表明文本包含更多不同的内容和词汇多样性。
2. 词汇选择和特征选择：信息熵可以帮助选择具有显著信息量的词汇或者特征，从而在文本分类、情感分析等任务中提高性能。
3. 发现异常文本：通过计算文本的信息熵，可以找出在语料库中具有较高或较低信息量的文本，这些文本可能具有特殊的特征或者潜在的异常。
4. 文本压缩：信息熵可以用来评估文本的压缩效率。较低的信息熵表示文本中的重复和冗余内容较多，可以通过压缩算法来减少数据存储空间。

具体代码如下：

```
# 计算主题的信息熵
topic_entropies = []
for topic in lda.show_topics():
    topic_words = [word for word, prob in lda.show_topic(topic[0])]
    word_probs = [prob for word, prob in lda.show_topic(topic[0])]
    entropy = np.sum(-np.array(word_probs) * np.log2(word_probs))
    topic_entropies.append((topic[0], topic_words, entropy))

z_data1 = []
x_data1 = []
y_data1 = []
# 打印主题及其对应的信息熵
for topic_entropy in topic_entropies:
    print("Topic {}: {}".format(topic_entropy[0], topic_entropy[1]))
    x_data1.append(topic_entropy[0])
    y_data1.append(topic_entropy[1])
    print("Entropy: {}".format(topic_entropy[2]))
    z_data1.append(topic_entropy[2])

#将上面获取的数据进行保存
df2 = pd.DataFrame()
```

```
df2['主题数'] = x_data1
df2['主题词'] = y_data1
df2['信息熵'] = z_data1
df2.to_csv('./result2/主题词与信息熵.csv',encoding='utf-8-sig',index=False)


def entropy(prob_list):
    # 初始化信息熵为0
    ent = 0
    # 遍历列表中的每个概率
    for p in prob_list:
        # 如果概率为0，跳过该项
        if p == 0:
            continue
        # 否则，用公式累加信息熵
        else:
            ent -= p * math.log2(p)
    # 返回信息熵
    return ent

def main1(text):
    # 对文本进行分词，用空格分隔
    words = text.split()

    # 统计每个词的出现次数，用字典存储
    word_count = {}
    for word in words:
        # 如果词已经在字典中，增加其计数
        if word in word_count:
            word_count[word] += 1
        # 否则，初始化其计数为1
        else:
            word_count[word] = 1

    # 计算每个词的概率，用列表存储
    prob_list = []
    # 获取文本的总词数
    total_words = len(words)
    # 遍历字典中的每个词和计数
    for word, count in word_count.items():
        # 计算词的概率，即计数除以总词数
        prob = count / total_words
        # 将概率添加到列表中
        prob_list.append(prob)

    # 调用之前定义的函数，计算信息熵
    ent = entropy(prob_list)
    return ent


df['entropy_values'] = df['clearn_comment'].apply(main1)
```

在处理好上面的所以内容后，我们需要去查看每日的指标他们的趋势如何

像每日，信息熵，评论数，点赞，转发这些的平均值如何，从而可以知道，一个社媒的整体舆情情况

这时，我们先对时间数据进行加工处理一下，接着再用groupby来对时间进行归类，从而求得平均值状况

具体代码如下：

```
def demo1():
    df = pd.read_csv('./result1/new_data.csv')
    def main1(x):
        x1 = str(x).split(" ")
        x2 = x1[0] + " "+ x1[1] + " "+ x1[2] + " " + x1[-1]
        return x2

    df['发文时间'] = df['发文时间'].apply(main1)
    df['发文时间'] = pd.to_datetime(df['发文时间'])
    new_df = df.groupby('发文时间').agg('mean')
    df1 = pd.DataFrame()
    df1['转发'] = new_df['转发']
    df1['点赞'] = new_df['点赞']
    df1['评论'] = new_df['评论']
    df1['熵值'] = new_df['entropy_values']

    plt.figure(figsize=(20,9),dpi=500)
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.plot(list(df1.index),df1['转发'], color='#A93226',linewidth=3)
    plt.title('retweet trend')
    plt.xlabel('DAY')
    plt.ylabel('value')
    plt.grid()
    plt.savefig('./result1/retweet trend.png')
    plt.show()


    plt.figure(figsize=(20,9),dpi=500)
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.plot(list(df1.index),df1['点赞'], color='#3498DB',linewidth=3)
    plt.title('like trend')
    plt.xlabel('DAY')
    plt.ylabel('value')
    plt.grid()
    plt.savefig('./result1/like trend.png')
    plt.show()

    plt.figure(figsize=(20,9),dpi=500)
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.plot(list(df1.index),df1['评论'], color='#F39C12',linewidth=3)
    plt.title('comment trend')
    plt.xlabel('DAY')
    plt.ylabel('value')
    plt.grid()
    plt.savefig('./result1/comment trend.png')
    plt.show()


    plt.figure(figsize=(20,9),dpi=500)
```

```
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.plot(list(df1.index),df1['熵值'], color='#2E4053',linewidth=3)
    plt.title('entropy trend')
    plt.xlabel('DAY')
    plt.ylabel('value')
    plt.grid()
    plt.savefig('./result1/entropy trend.png')
    plt.show()

    df1.to_csv('./result1/时间数据.csv',encoding='utf-8-sig')
```

以上便是全部内容了