Cleaning SVM Bag of Words Question

In [1]:
```
from string import punctuation
from os import listdir
from collections import Counter
from nltk.corpus import stopwords
import string
from keras.preprocessing.text import Tokenizer
import numpy as np
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
```

- nltk is a great library for NLP that we can use to get items such as a list of common stop words as you will see below
- keras does a good job making it easy for us to set up the bag of words model easily by simply feeding in text rather than implementing it ourselves as we did in the first HW question
- sklearn is a library we use to simply build ML models we have discussed in this class such as SVMs and Logistic Regression. It will alsoo allow us to see the confusion matrix and get accuracy easily.

1. Cleaning data

For this question we use IMBD movie reviews from Kaggle and attempt to use SVM (which we learned about earlier in this class) and the bag of words model to do sentiment analysis. We also examine the effect of data cleaning on train/test accuracy.

Part A: Identifying and Adding Cleaning functions

We first get an example review and attempt to create a function to clean the data. Below Identify what the three given cleaning sections do and explain why they are helpful, and write code for a fourth section that would aid in removing words such as "I" or "A" which do not have an impact on sentiment analysis.

Hint: Consider the minimum length of useful information

In [2]:
```
# function for getting the doc
def get_doc(filename):
    f = open(filename, 'r')
    txt = f.read()
    f.close()
    return txt

# Used if we did not clean file
def not_clean_file(f):
    data = f.split()
    return data

# function for cleaning the doc
def clean_file(f):
    # we grab all the data seperated by whitespace
    data = f.split()

    # Clean 1
    table = str.maketrans('', '', string.punctuation)
    data = [w.translate(table) for w in data]

    #  Clean 2
    data = [w for w in data if w.isalpha()]

    # Clean 3
    # Recall from our notes what stop words are. Here we use the stopwords
    # to a list of common stop words in english. EX. words include:
    #{‘ourselves’, ‘hers’, ‘between’, ‘yourself’, ‘but’, ‘again’, ‘there’,
    stop_words = set(stopwords.words('english'))
    data = [w for w in data if not w in stop_words]

    ### Begin Part A
    data = [word for word in data if len(word) > 1]
    ### End Part A
    return data


# get the cleaned text
f = 'data/pos/cv000_29590.txt'
text = get_doc(f)
print("Original text: ")
print(text[:1000])

cleaned_text = clean_file(text)
not_cleaned_text = not_clean_file(text)

print()
print("Words from text: ")
print(not_cleaned_text[:30])

print()
print("Cleaned words from text: ")
print(cleaned_text[:20])
```

```
Original text:
films adapted from comic books have had plenty of success , whether the
```

```
y're about superheroes ( batman , superman , spawn ) , or geared toward
kids ( casper ) or the arthouse crowd ( ghost world ) , but there's nev
er really been a comic book like from hell before .
for starters , it was created by alan moore ( and eddie campbell ) , wh
o brought the medium to a whole new level in the mid '80s with a 12-par
t series called the watchmen .
to say moore and campbell thoroughly researched the subject of jack the
ripper would be like saying michael jackson is starting to look a littl
e odd .
the book ( or " graphic novel , " if you will ) is over 500 pages long
and includes nearly 30 more that consist of nothing but footnotes .
in other words , don't dismiss this film because of its source .
if you can get past the whole comic book thing , you might find another
stumbling block in from hell's directors , albert and allen hughes .
getting the hughes brothers to direct this seems almost as

Words from text:
['films', 'adapted', 'from', 'comic', 'books', 'have', 'had', 'plenty',
'of', 'success', ',', 'whether', "they're", 'about', 'superheroes',
'(', 'batman', ',', 'superman', ',', 'spawn', ')', ',', 'or', 'geared',
'toward', 'kids', '(', 'casper', ')']

Cleaned words from text:
['films', 'adapted', 'comic', 'books', 'plenty', 'success', 'whether',
'theyre', 'superheroes', 'batman', 'superman', 'spawn', 'geared', 'towa
rd', 'kids', 'casper', 'arthouse', 'crowd', 'ghost', 'world']
```

RESPONSE:

In [3]:
```
# Clean 1:
# We first remove punctuation from each token. This is useful since
# punctuation would get added at the end of strings making them seem
# like unique words while they are not.
# Clean 2:
# We remove any words that are not alphabetic. We assume that symbols and
# numbers do not contribute as much to sentiment analysis and remove them.
# Clean 3:
# We filter out stop words. If we pring out the stop words, we see they
# are a list of words that do not add sentiment value and thus can be
# removed without impact
```

2. Training without Cleaning

Part B: Building Vocabulary

We now create a vocabulary that we can use for later steps. To do this we run the functions from before for all the train data. For this part of the assignment we wil NOT be cleaning data.

In [4]:
```
# load doc and add to vocab (not clean)
def add_doc_to_vocab(filename, vocab):
    doc = get_doc(filename)
    not_cleaned = not_clean_file(doc)
    vocab.update(not_cleaned)


def process_docs(directory, vocab):
    for filename in listdir(directory):
        # skip any reviews in the test set
        if filename.startswith('cv9'):
            continue
        path = directory + '/' + filename
        add_doc_to_vocab(path, vocab)

# define vocab as a counter type
vocab = Counter()
# Adding both positive and negative data
process_docs('data/pos', vocab)
process_docs('data/neg', vocab)
# Printing the most common words from vocab
print(vocab.most_common(50))
```

```
[(',', 69706), ('the', 68273), ('.', 59103), ('a', 34138), ('and', 3164
2), ('of', 30419), ('to', 28503), ('is', 22501), ('in', 19410), ('"', 157
98), ('that', 13536), ('it', 11068), (')', 10577), ('(', 10467), ('as', 1
0175), ('with', 9651), ('for', 8880), ('his', 8602), ('this', 8563), ('fi
lm', 7974), ('but', 7727), ('he', 6816), ('i', 6710), ('on', 6479), ('ar
e', 6232), ('by', 5608), ('be', 5468), ('an', 5099), ('one', 4939), ('no
t', 4913), ('who', 4859), ('movie', 4815), ('at', 4464), ('was', 4420),
('from', 4417), ('have', 4400), ('has', 4266), ('you', 4010), ('her', 396
3), ('they', 3849), ('all', 3819), ('?', 3397), ("it's", 3348), ('so', 32
38), ('like', 3193), ('about', 3141), ('out', 3071), ('more', 2991), ('wh
en', 2957), ('which', 2849)]
```

What do you notice about the most common values in the vocabulary above. Do you think that they are helpful in our sentiment analysis?

RESPONSE:

In [5]:
```
# We notice that the most common words seem to be punctuation, one letter
# words and other words such as the and a that do not add value to our
# sentiment analysis. This is why we do cleaning on our data.
```

Part C: removing values that appear less than once

We do not need to include words that appear only once in our vocabulary as they are most likely unique words that are not common and do not play a major role in sentiment analysis. Write the ccode below to remove all words with length less than 5.

In [6]:
```
### Start Part C
min_occurance = 5
trim_vocab = [k for k,c in vocab.items() if c >= min_occurance]
### End Part C
```

We then save this vocab as a file to use for later.

In [7]:
```
def save_file(lines, filename):
    data = '\n'.join(lines)
    file = open(filename, 'w')
    file.write(data)
    file.close()

# save tokens to a vocabulary file
save_file(trim_vocab, 'vocab_unclean.txt')
```

Part D: Creating a model

We create helper functions that will allow us to properly use SVMs as our learning models. Please complete the code segments below.

In [8]:
```
# Function we will use to load a doc and grab all values that are also
# in vocab
def doc_to_line(filename, vocab):
    doc = get_doc(filename)
    words = not_clean_file(doc)
    # Write code to only include words that are in the vocabulary
    ### Begin Part D
    words = [w for w in words if w in vocab]
    ### End Part D
    return ' '.join(words)
```

In [9]:
```
# Loads all data given whether it is train or test
def process_docs(directory, vocab, is_trian):
    lines = list()
    for filename in listdir(directory):
        # choose train or test data
        if is_trian and filename.startswith('cv9'):
            continue
        if not is_trian and not filename.startswith('cv9'):
            continue

        path = directory + '/' + filename
        line = doc_to_line(path, vocab)
        lines.append(line)
    return lines
```

In [10]:
```
# We use tokenizer in order to generate our Xtrain and Xtest
# This uses the Tokenizer library in order to help create the featurization
# input words and vocabulary we have.

# the Tokenizer provides 4 attributes that you can use to query what has be

# word_counts: A dictionary of words and their counts.
# word_docs: A dictionary of words and how many documents each appeared in.
# word_index: A dictionary of words and their uniquely assigned integers.
# document_count:An integer count of the total number of documents that wer

# This function provides a suite of standard bag-of-words model text encodi
# via a mode argument to the function.

# The modes available include:

# ‘binary‘: Whether or not each word is present in the document. This is th
# ‘count‘: The count of each word in the document.
# ‘tfidf‘: The Text Frequency-Inverse DocumentFrequency (TF-IDF) scoring fo
# ‘freq‘: The frequency of each word as a ratio of words within each docume

# For more information about Tokenizer, you can visit:
# https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/

# View this link: https://keras.io/api/preprocessing/text/#tokenizer
# For documentation information
def prepare_data(train_docs, test_docs, mode):
    # We create the tokenizer
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(train_docs)
    # encode train data set
    Xtrain = tokenizer.texts_to_matrix(train_docs, mode=mode)
    # encode test data set
    Xtest = tokenizer.texts_to_matrix(test_docs, mode=mode)
    return Xtrain, Xtest
```

Part E: Running the Model

We now begin to run the model. Please complete the code below and answer the following questions.

In [11]:
```
# load the vocabulary
vocab_filename = 'vocab_unclean.txt'
vocab = get_doc(vocab_filename)
vocab = vocab.split()
vocab = set(vocab)
```

In [12]:
```
# load all training reviews
positive_lines = process_docs('data/pos', vocab, True)
negative_lines = process_docs('data/neg', vocab, True)
train_docs = negative_lines + positive_lines
```

In [13]:
```
positive_lines = process_docs('data/pos', vocab, False)
negative_lines = process_docs('data/neg', vocab, False)
test_docs = negative_lines + positive_lines
```

In [14]:
```
# prepare labels
ytrain = np.array([0 for _ in range(900)] + [1 for _ in range(900)])
ytest = np.array([0 for _ in range(100)] + [1 for _ in range(100)])
```

In [15]:
```
# Note here we use the binary option. If you read the explanation for Token
# Feel free to try 'count', 'freq', or 'tfidf' instead, but we do not requi
Xtrain, Xtest = prepare_data(train_docs, test_docs, 'binary')
# Write code below to use SVMs to create a model. You may use sklearn. You
# https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
### Begin Part E
clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
clf.fit(Xtrain, ytrain)
### End Part E
```

Out[15]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc', SVC(gamma='auto'))])
```

In [16]:
```
# Print train error
### Begin Part E
clf.score(Xtrain, ytrain)
### End Part E
```

Out[16]:
```
0.9922222222222222
```

In [17]:
```
# Print test error
### Begin Part E
clf.score(Xtest, ytest)
### End Part E
```

Out[17]:
```
0.835
```

In [18]:
```
# Print Confusion Matrix
### Begin Part E
ypred = clf.predict(Xtest)
print(confusion_matrix(ytest, ypred))
### End Part E
```

```
[[91  9]
 [24 76]]
```

How well did your model perform. Is it what you expected?

RESPONSE:

In [19]:
```
# Any answer here with sufficient thought will suffice. EX: The train and
# test error were expected given this model because ...
```

Part F: Vocabulary with Clean Data

We will recreate our vocabulary but with cleaned data this data. Respond to the question below.

In [20]:
```
# load doc and add to vocab (clean)
def add_doc_to_vocab2(filename, vocab):
    doc = get_doc(filename)
    cleaned = clean_file(doc)
    vocab.update(cleaned)

def process_docs2(directory, vocab):
    for filename in listdir(directory):
        # skip any reviews in the test set
        if filename.startswith('cv9'):
            continue
        path = directory + '/' + filename
        add_doc_to_vocab2(path, vocab)

# define vocab as a counter type
vocab2 = Counter()
# Adding both positive and negative data
process_docs2('data/pos', vocab2)
process_docs2('data/neg', vocab2)
# Printing the most common words from vocab
print(vocab2.most_common(50))
```

```
[('film', 7983), ('one', 4946), ('movie', 4826), ('like', 3201), ('even',
2262), ('good', 2080), ('time', 2041), ('story', 1907), ('films', 1873),
('would', 1844), ('much', 1824), ('also', 1757), ('characters', 1735),
('get', 1724), ('character', 1703), ('two', 1643), ('first', 1588), ('se
e', 1557), ('way', 1515), ('well', 1511), ('make', 1418), ('really', 140
7), ('little', 1351), ('life', 1334), ('plot', 1288), ('people', 1269),
('could', 1248), ('bad', 1248), ('scene', 1241), ('movies', 1238), ('neve
r', 1201), ('best', 1179), ('new', 1140), ('scenes', 1135), ('man', 113
1), ('many', 1130), ('doesnt', 1118), ('know', 1092), ('dont', 1086), ('h
es', 1024), ('great', 1014), ('another', 992), ('action', 985), ('love',
977), ('us', 967), ('go', 952), ('director', 948), ('end', 946), ('someth
ing', 945), ('still', 936)]
```

How do the most common words compare to that of Part B when we built the vocabulary without cleaning?

RESPONSE:

In [21]:
```
# The words are much more relevant to sentiment analysis. We do not see
# punctuation anymore or words that dont have meaning to sentiment.
```

Re-add the code from Part C below to remove values that appear less than five times.

In [22]:
```
### Start Part F
min_occurance = 5
trim_vocab2 = [k for k,c in vocab2.items() if c >= min_occurance]
### End Part F
```

In [23]:
```
# save tokens to a vocabulary file
save_file(trim_vocab2, 'vocab_clean.txt')
```

Part G: Training with Clean Data

We re-train the model with clean data this time. Add code below and answer the following questions.

In [24]:
```
# Function we will use to load a doc and grab all values that are also
# in vocab
def doc_to_line2(filename, vocab):
    doc = get_doc(filename)
    words = clean_file(doc)
    # Write code to only include words that are in the vocabulary
    # This is the same as part D
    ### Begin Part G
    words = [w for w in words if w in vocab]
    ### End Part G
    return ' '.join(words)

# Loads all data given whether it is train or test
def process_docs(directory, vocab, is_trian):
    lines = list()
    for filename in listdir(directory):
        # choose train or test data
        if is_trian and filename.startswith('cv9'):
            continue
        if not is_trian and not filename.startswith('cv9'):
            continue

        path = directory + '/' + filename
        line = doc_to_line2(path, vocab)
        lines.append(line)
    return lines
```

In [25]:
```
# load the vocabulary
vocab_filename = 'vocab_clean.txt'
vocab = get_doc(vocab_filename)
vocab = vocab.split()
vocab = set(vocab)
```

In [26]:
```
# load all training reviews
positive_lines = process_docs('data/pos', vocab, True)
negative_lines = process_docs('data/neg', vocab, True)
train_docs = negative_lines + positive_lines
```

In [27]:
```
positive_lines = process_docs('data/pos', vocab, False)
negative_lines = process_docs('data/neg', vocab, False)
test_docs = negative_lines + positive_lines
```

In [28]:
```
# prepare labels
ytrain = np.array([0 for _ in range(900)] + [1 for _ in range(900)])
ytest = np.array([0 for _ in range(100)] + [1 for _ in range(100)])
```

In [29]:
```
Xtrain, Xtest = prepare_data(train_docs, test_docs, 'binary')
# Write code below to use SVMs to create a model. You may use sklearn
# Same as Part E
### Begin Part G
clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
clf.fit(Xtrain, ytrain)
### End Part G
```

Out[29]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc', SVC(gamma='auto'))])
```

In [30]:
```
# Print train error
# Same as Part E
### Begin Part G
clf.score(Xtrain, ytrain)
### End Part G
```

Out[30]:
```
0.9911111111111112
```

In [31]:
```
# Print test error
# Same as Part E
### Begin Part G
clf.score(Xtest, ytest)
### End Part G
```

Out[31]:
```
0.865
```

In [32]:
```
# Print Confusion Matrix
# Same as Part E
### Begin Part G
ypred = clf.predict(Xtest)
print(confusion_matrix(ytest, ypred))
### End Part G
```

```
[[91  9]
 [18 82]]
```

Did you expect these results? What effect did cleaning the data before training have?

RESPONSE:

In [33]:
```
# Any thoughtful answer that explores the impact of cleaning on reducing th
# it has is sufficient
```