BANA 275: NATRL LANG PROCESS

Homework 1: Rule-Based Classification

The first programming assignment will familiarize you with the basic text processing methods, the use of pre-built lexicons and rules for text classification.

1 Task: Sentiment Classification

The primary objective for the assignment is to predict the sentiment of a movie review. In particular, we will be providing you with a dataset containing the text of the movie reviews from IMDB, and for each review, you have to predict whether the review is positive or negative. We will also provide some code to help you read and write the output files.

1.1 Data

The primary data file is named data.zip, which contains the following:

```
- lexicon/: Two sentiment lexicons. The code for reading them is included.
- test/: Folder of text files containing reviews that are not labeled.
- train/: Folder of text files containing the reviews that are part of labeled dat
a.
- train.csv: List of files and associated sentiment label, for evaluating your clas
sifier.

Note: train/ and test/ folder should contain 25,000 files. If you have 25,001 on di
sk, remember to delete the .DS_Store or desktop.ini before running any code.
```

1.2 Kaggle

Kaggle is a website that hosts machine learning competitions, and we will be using it to evaluate and compare the accuracy of your classifiers. We know the true sentiment for each of the *unlabeled* reviews, which we will use to evaluate your submissions, and thus your submission file to Kaggle should contain a predicted label for all the unlabeled reviews. In particular, the submission file `test.csv` should have the following format (code already does this):

- Start with a single line header: `Fileindex, Category`
- For each of the unlabeled speech (sorted by name) there is a line containing an increasing integer index (i.e. line number 1), then a comma, and then the string label prediction of that speech.
- See `test-basic.csv` for example.

You can make ***at most three*** submissions each day, so we encourage you to test your submission files early, and observe the performance of your system. By the end of the submission period, you will have to select the two submissions the best of which you want to be judged as your final submission. Public leaderboard uses 30% of the data while your performance is evaluated by private leaderboard that uses 70% of the data.

1.3 Source Code

Some initial code contains methods for loading the data and lexicons, and calling the methods to run and evaluate your classifier. It also contains the code to output the submission file from your classifier (called `test.csv`) that you will submit to Kaggle. Your directory structure should look like this.

```
hw1
|
└──code
|   └──rule-based-classification.ipynb
└──data
|   └──lexicon
|       |   inqtabs.txt
|       └──SentiWordNet_3.0.0_20130122.txt
|   └──test
|       |   0.txt
|       |   1.txt
|       |   ...
|       └──24999.txt
|   └──train
|       |   0.txt
|       |   1.txt
|       |   ...
|       └──24999.txt
|   └──train.csv
└──output
    |   fn.txt
    |   fp.txt
    |   test.csv
    |   tn.txt
    └──tp.txt
Note: You need mannually create folder 'code' and 'output'.
```

This code block contains the skeleton of your classifier; this is the **only** part you need to modify.

2 What to submit?

Prepare and submit a single write-up (**PDF, maximum 2 pages**) and a jupyter notebook to Canvas. **Do not include your student ID number** , since we might share it with the class if it's worth highlighting. The write-up and code should address the following.

2.1 Preliminaries (5 points)

At the top of your write-up, include your team's Kaggle name such as '**Sec A Team 1**' , and the accuracy that your ***best*** submission obtained on Kaggle. You do **not** need to include any other details such as name, UCINet Id, etc.

2.2 Rule-Based Classifier (40 points)

Your main goal is to improve the basic classifier. For this, you should consider doing both of the following:

- **Lexicons** : We have provided two lexicons for your use. Each lexicon is a dictionary containing words as keys and the sentiment as the value. For Harvard Inquirer (inqtabs_dict) (http://www.wjh.harvard.edu/~inquirer/), the value is a sentiment label: 0 for negative and 1 for positive. For SentiWordNet (swn_dict) (http://sentiwordnet.isti.cnr.it/), each value is a pair of positive and negative scores, respectively. Use them as you see fit.
- **Regular Expressions** : After looking at some reviews, you may have ideas for rules on the review text that you think will help predict the sentiment. Implement them using if/then and regular expressions.

Implement your suggestions in `classify()` , and describe them in a few sentences in your report. The primary evaluation for this part will be the performance of your classifier, combined with how creative/interesting your proposed ideas are.

2.3 Examples (30 points)

In order to aid analysis, you also need to figure out the errors being made by your classifiers, i.e. split each prediction into *four* categories: true positives, true negatives, false positives, and false negatives. If you look at `get_error_type()` , there is an incorrect implementation of this method. Fix this code to print the appropriate examples, which will result in 4 files full of reviews, called `fp.txt, fn.txt, tp.txt, and tn.txt` . Include 2-3 examples from the false positives and negatives in your report.

2.4 Analysis (20 points)

Analyze the above false positive and false negatives in your writeup. In particular, in a few sentences, describe what is lacking in your approach, i.e. why do you think the errors exist. Write a sentence or two about how you would address them if you had more time. You will be evaluated on how well you were able to identify the problems, and the creativity of your proposed future solution.

3 Statement of Collaboration (5 points)

It is **mandatory** to include a *Statement of Collaboration* in each submission, with respect to the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments, in particular, we encourage the students to organize to discuss the task descriptions, requirements, bugs in our code, and the relevant technical content *before* they start working on it. However, you should not discuss the specific solutions, and, as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (i.e. no photographs of the blackboard, written notes, etc.). Especially *after* you have started working on the assignment, try to restrict the discussion on Canvas as much as possible,

In [1]:
```
# Some initial codes. Do not modify.
import os
import csv
import sys
from tqdm.notebook import tqdm

POS_LABEL = '1'
NEG_LABEL = '0'


def check_if_exist(file_path):
    if not os.path.exists(file_path):
        print(file_path + ' could not be found')
        return False
    return True


def extract_word(word):
    return word.lower() if word.find('#') < 0 else word[:word.find('#')].lower
()


def read_inqtabs(input_file_path):
    """
    :param input_file_path:
    :return lexicons: dictionary of labels (e.g. lexicons['good']: 1, lexicons
['bad']: 0)
    """
    if not check_if_exist(input_file_path):
        return

    lexicons = dict()
    with open(input_file_path, 'r', encoding='utf-8') as fp:
        for line in fp.readlines():
            elements = line.strip().split('\t')
            word = extract_word(elements[0])
            if len(word) > 0 and (elements[2] == 'Positiv' or elements[3] ==
'Negativ'):
                label = POS_LABEL if elements[2] == 'Positiv' else NEG_LABEL
                lexicons[word] = label
    return lexicons


def read_senti_word_net(input_file_path):
    """
    :param input_file_path:
    :return lexicon: dictionary of lists (e.g. lexicons['good'][0]: positive s
core, lexicons['bad'][1]: negative score)
    """
    if not check_if_exist(input_file_path):
        return

    all_lexicons = dict()
    with open(input_file_path, 'r', encoding='utf-8') as fp:
        for line in fp.readlines():
            if line.startswith('#'):
```

```
                continue

            elements = line.strip().split('\t')
            if len(elements) < 5 or len(elements[4]) == 0:
                continue

            for tmp_word in elements[4].split(' '):
                word = extract_word(tmp_word).replace('_', ' ')
                if len(word) > 0 and len(elements[2]) > 0 and len(elements[3])
> 0:
                    if word not in all_lexicons.keys():
                        all_lexicons[word] = list()
                        all_lexicons[word].append(list())
                        all_lexicons[word].append(list())
                    all_lexicons[word][0].append(float(elements[2]))
                    all_lexicons[word][1].append(float(elements[3]))

    lexicons = dict()
    for word in all_lexicons.keys():
        lexicons[word] = (max(all_lexicons[word][0]), max(all_lexicons[word][1
]))
    return lexicons


def get_training_data(filedir):
    with open(os.path.join(filedir, 'train.csv'), encoding='utf-8') as csvfile
:
        training_data = [row for row in csv.DictReader(csvfile, delimiter=','
)]
        for entry in training_data:
            with open(os.path.join(filedir, 'train', entry['FileIndex'] + '.tx
t'), encoding='utf-8') as reviewfile:
                entry['Review'] = reviewfile.read()
    return training_data


def get_training_accuracy(data, inqtabs_dict, swn_dict):
    num_correct = 0
    etype_files = {}
    for etype in ["fp", "fn", "tp", "tn"]:
        etype_files[etype] = open('../output/'+ etype + '.txt', 'w+', encoding
='utf-8')
    for row in data:
        sentiment_prediction = classify(row['Review'], inqtabs_dict, swn_dict)
        sentiment_label = int(row['Category'])
        if sentiment_prediction == sentiment_label:
            num_correct += 1
        etype = get_error_type(sentiment_prediction, sentiment_label)
        etype_files[etype].write("%s\t%s\n"%(row['FileIndex'], row['Review']))
    accuracy = num_correct * 1.0 / len(data)
    for etype in ["fp", "fn", "tp", "tn"]:
        etype_files[etype].close()
    print("Accuracy: " + str(accuracy))
    return accuracy


def write_predictions(filedir, inqtabs_dict, swn_dict, output_file_name):
```

```
    testfiledir = os.path.join(filedir, 'test')
    with open(output_file_name, 'w+', encoding='utf-8') as csvfile:
        writer = csv.DictWriter(csvfile, delimiter=',', fieldnames=['FileInde
x', 'Category'])
        writer.writeheader()
        for filename in tqdm(sorted(os.listdir(testfiledir), key=lambda x: int
(os.path.splitext(x)[0]))):
            with open(os.path.join(testfiledir, filename), encoding='utf-8') a
s reviewfile:
                review = reviewfile.read()
                prediction = dict()
                prediction['FileIndex'] = os.path.splitext(filename)[0]
                prediction['Category'] = classify(review, inqtabs_dict, swn_di
ct)
                writer.writerow(prediction)
```

In [2]:
```
def get_training_accuracy(data, inqtabs_dict, swn_dict):
    num_correct = 0
    etype_files = {}
    for etype in ["fp", "fn", "tp", "tn"]:
        etype_files[etype] = open('../output/'+ etype + '.txt', 'w+', encoding
='utf-8')
    for row in tqdm(data):
        sentiment_prediction = classify(row['Review'], inqtabs_dict, swn_dict)
        sentiment_label = int(row['Category'])
        if sentiment_prediction == sentiment_label:
            num_correct += 1
        etype = get_error_type(sentiment_prediction, sentiment_label)
        etype_files[etype].write("%s\t%s\n"%(row['FileIndex'], row['Review']))
    accuracy = num_correct * 1.0 / len(data)
    for etype in ["fp", "fn", "tp", "tn"]:
        etype_files[etype].close()
    print("Accuracy: " + str(accuracy))
    return accuracy
```

Code Block for you to modify:

In [3]:
```
import nltk
from nltk.corpus import stopwords
import re
from nltk.tokenize import word_tokenize
stop_words = set(stopwords.words('english'))
from nltk.corpus import wordnet as wn
```

In [4]:
```
#Removes Punctuations
def remove_punctuations(data):
    punct_tag=re.compile(r'[^\w\s]')
    data=punct_tag.sub(r'',data)
    return data

#Removes HTML syntaxes
def remove_html(data):
    html_tag=re.compile(r'<.*?>')
    data=html_tag.sub(r'',data)
    return data

def remove_url(data):
    url_clean= re.compile(r"https://\S+|www\.\S+")
    data=url_clean.sub(r'',data)
    return data

def remove_emoji(data):
    emoji_clean= re.compile("["
                            u"\U0001F600-\U0001F64F"  # emoticons
                            u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                            u"\U0001F680-\U0001F6FF"  # transport & map symbols
                            u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                            u"\U00002702-\U000027B0"
                            u"\U000024C2-\U0001F251"
                            "]+", flags=re.UNICODE)
    data=emoji_clean.sub(r'',data)
    url_clean= re.compile(r"https://\S+|www\.\S+")
    data=url_clean.sub(r'',data)
    return data

def remove_abb(data):
    data = re.sub(r"he's", "he is", data)
    data = re.sub(r"there's", "there is", data)
    data = re.sub(r"We're", "We are", data)
    data = re.sub(r"That's", "That is", data)
    data = re.sub(r"won't", "will not", data)
    data = re.sub(r"they're", "they are", data)
    data = re.sub(r"Can't", "Cannot", data)
    data = re.sub(r"wasn't", "was not", data)
    data = re.sub(r"don\x89Ûªt", "do not", data)
    data= re.sub(r"aren't", "are not", data)
    data = re.sub(r"isn't", "is not", data)
    data = re.sub(r"What's", "What is", data)
    data = re.sub(r"haven't", "have not", data)
    data = re.sub(r"hasn't", "has not", data)
    data = re.sub(r"There's", "There is", data)
    data = re.sub(r"He's", "He is", data)
    data = re.sub(r"It's", "It is", data)
    data = re.sub(r"You're", "You are", data)
    data = re.sub(r"I'M", "I am", data)
    data = re.sub(r"shouldn't", "should not", data)
    data = re.sub(r"wouldn't", "would not", data)
    data = re.sub(r"i'm", "I am", data)
    data = re.sub(r"I\x89Ûªm", "I am", data)
    data = re.sub(r"I'm", "I am", data)
```

```
data = re.sub(r"Isn't", "is not", data)
data = re.sub(r"Here's", "Here is", data)
data = re.sub(r"you've", "you have", data)
data = re.sub(r"you\x89Ûªve", "you have", data)
data = re.sub(r"we're", "we are", data)
data = re.sub(r"what's", "what is", data)
data = re.sub(r"couldn't", "could not", data)
data = re.sub(r"we've", "we have", data)
data = re.sub(r"it\x89Ûªs", "it is", data)
data = re.sub(r"doesn\x89Ûªt", "does not", data)
data = re.sub(r"It\x89Ûªs", "It is", data)
data = re.sub(r"Here\x89Ûªs", "Here is", data)
data = re.sub(r"who's", "who is", data)
data = re.sub(r"I\x89Ûªve", "I have", data)
data = re.sub(r"y'all", "you all", data)
data = re.sub(r"can\x89Ûªt", "cannot", data)
data = re.sub(r"would've", "would have", data)
data = re.sub(r"it'll", "it will", data)
data = re.sub(r"we'll", "we will", data)
data = re.sub(r"wouldn\x89Ûªt", "would not", data)
data = re.sub(r"We've", "We have", data)
data = re.sub(r"he'll", "he will", data)
data = re.sub(r"Y'all", "You all", data)
data = re.sub(r"Weren't", "Were not", data)
data = re.sub(r"Didn't", "Did not", data)
data = re.sub(r"they'll", "they will", data)
data = re.sub(r"they'd", "they would", data)
data = re.sub(r"DON'T", "DO NOT", data)
data = re.sub(r"That\x89Ûªs", "That is", data)
data = re.sub(r"they've", "they have", data)
data = re.sub(r"i'd", "I would", data)
data = re.sub(r"should've", "should have", data)
data = re.sub(r"You\x89Ûªre", "You are", data)
data = re.sub(r"where's", "where is", data)
data = re.sub(r"Don\x89Ûªt", "Do not", data)
data = re.sub(r"we'd", "we would", data)
data = re.sub(r"i'll", "I will", data)
data = re.sub(r"weren't", "were not", data)
data = re.sub(r"They're", "They are", data)
data = re.sub(r"Can\x89Ûªt", "Cannot", data)
data = re.sub(r"you\x89Ûªll", "you will", data)
data = re.sub(r"I\x89Ûªd", "I would", data)
data = re.sub(r"let's", "let us", data)
data = re.sub(r"it's", "it is", data)
data = re.sub(r"can't", "cannot", data)
data = re.sub(r"don't", "do not", data)
data = re.sub(r"you're", "you are", data)
data = re.sub(r"i've", "I have", data)
data = re.sub(r"that's", "that is", data)
data = re.sub(r"i'll", "I will", data)
data = re.sub(r"doesn't", "does not",data)
data = re.sub(r"i'd", "I would", data)
data = re.sub(r"didn't", "did not", data)
data = re.sub(r"ain't", "am not", data)
data = re.sub(r"you'll", "you will", data)
data = re.sub(r"I've", "I have", data)
data = re.sub(r"Don't", "do not", data)
```

```
    data = re.sub(r"I'll", "I will", data)
    data = re.sub(r"I'd", "I would", data)
    data = re.sub(r"Let's", "Let us", data)
    data = re.sub(r"you'd", "You would", data)
    data = re.sub(r"It's", "It is", data)
    data = re.sub(r"Ain't", "am not", data)
    data = re.sub(r"Haven't", "Have not", data)
    data = re.sub(r"Could've", "Could have", data)
    data = re.sub(r"youve", "you have", data)
    data = re.sub(r"donå«t", "do not", data)
    return data
```

In [5]:
```
from nltk.stem import WordNetLemmatizer

def lemma_traincorpus(data):
    lemmatizer=WordNetLemmatizer()
    out_data=""
    for words in data:
        out_data+= lemmatizer.lemmatize(words)
    return out_data
```

In [6]:
```
def penn_to_wn(tag):
    if tag.startswith('J'):
        return wn.ADJ
    elif tag.startswith('N'):
        return wn.NOUN
    elif tag.startswith('R'):
        return wn.ADV
    elif tag.startswith('V'):
        return wn.VERB
    return None
```

In [7]:
```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()
```

In [8]:
```
def get_error_type(pred, label):
    # return the type of error: tp,fp,tn,fn
    if pred == 1:
        if label == 1:
            return "tp"
        else:
            return "fp"
    if pred == 0:
        if label == 1:
            return "fn"
        else:
            return "tn"


import re

#Put the code under the classify method here.  Here's some code to start you o
ff
def classify(text, inqtabs_dict, swn_dict):

    text = re.sub('\[[^]]*\]', '', text)
    text = re.sub(r'\([^)]*\)', '', text)
    text = lemma_traincorpus(text)
    regex = r'[A-Za-z0-9]+' #You'll probably want to update this regular expre
ssion
    words = re.findall(regex, text)

    postagging = []

    words = [w for w in words if not w in stop_words]

    postagging.append(nltk.pos_tag(word_tokenize(" ".join(words))))


    positive_words_inqtabs = []
    negative_words_inqtabs = []
    words_matched_inqtabs = []

    positive_words_swn = {}
    negative_words_swn = {}
    words_matched_swn = []

    for word,y in postagging[0]:

        wn_tag = penn_to_wn(y)

        if wn_tag is None:

            continue
        else:
            if word in inqtabs_dict:
                if inqtabs_dict[word] == '1':
                    positive_words_inqtabs.append(word)
                else:
                    negative_words_inqtabs.append(word)
                words_matched_inqtabs.append(word)
```

```
            if word in swn_dict:
                positive_words_swn[word] = swn_dict[word][0]
                negative_words_swn[word] = swn_dict[word][1]
                words_matched_swn.append(word)

    #You'll definitely want to experiment with different ideas for the code be
low

    if len(positive_words_inqtabs) - len(negative_words_inqtabs)*1.3 > 0:
        score_i = 1
    else:
        score_i = 0

    if sum(positive_words_swn.values()) - sum(negative_words_swn.values())*1.1
75 > 0:

        score_s = 1
    else:
        score_s = 0
    if score_i== score_s:
        score = score_i
    else:
        score = 0

    return score
```

In [11]:
```
filedir = '../data'
output_file_name = '../output/test.csv'
print("Reading data...")
data = get_training_data(filedir)
lexicon_dir = os.path.join(filedir, 'lexicon')
inqtabs_dict = read_inqtabs(os.path.join(lexicon_dir, 'inqtabs.txt'))
swn_dict = read_senti_word_net(os.path.join(lexicon_dir, 'SentiWordNet_3.0.0_2
0130122.txt'))
print("Classifying...")
get_training_accuracy(data, inqtabs_dict, swn_dict)
print("Writing output...")
write_predictions(filedir, inqtabs_dict, swn_dict, output_file_name)
```

```
Reading data...
Classifying...


Accuracy: 0.70044
Writing output...
```

Code for Error Analysis

In [31]:
```
# Rerun this as you make changes to the classify method
# I strongly recommend you look at the number of false positives and false neg
atives after running this.
# You can see the false positives and false negatives by looking at the size o
f the fp.txt and fn.txt
# Note: you need to complete the get_error_type function to see the false posi
tives and false negatives

get_training_accuracy(data, inqtabs_dict, swn_dict)
# fp 11.6MB   fn 1.6 MB
```

```
Accuracy: 0.68236
```

Out[31]:
```
0.68236
```

Error Analysis, debugging classify on a single example

In [19]:
```
import re

#Put the code under the classify method here.  Here's some code to start you o
ff
def classify(text, inqtabs_dict, swn_dict):

    text = re.sub('\[[^]]*\]', '', text)
    text = re.sub(r'\([^)]*\)', '', text)
    regex = r'[A-Za-z0-9]+' #You'll probably want to update this regular expre
ssion
    words = re.findall(regex, text)

    words = [w for w in words if not w in stop_words]

    sntmnt = analyzer.polarity_scores(" ".join(words))['compound']

    #if sntmnt>0:
    if sntmnt>=0:
        score = 1
    else:
        score = 0

    return score
```

Vader

In [11]:
```
get_training_accuracy(data, inqtabs_dict, swn_dict)
```

```
Accuracy: 0.67712
```

Out[11]:
```
0.67712
```

In [20]:
```
# if sntmnt>=0:
get_training_accuracy(data, inqtabs_dict, swn_dict)
```

```
Accuracy: 0.67672
```

Out[20]:
```
0.67672
```

In [27]:
```
# Modify this to see it on a different example
text = '''
Arguably this is a very good "sequel", better than the first live action film
 101 Dalmatians. It has good dogs, good actors, good jokes and all right slaps
tick! <br /><br />Cruella DeVil, who has had some rather major therapy, is now
a lover of dogs and very kind to them. Many, including Chloe Simon, owner of o
ne of the dogs that Cruella once tried to kill, do not believe this. Others, l
ike Kevin Shepherd (owner of 2nd Chance Dog Shelter) believe that she has chan
ged. <br /><br />Meanwhile, Dipstick, with his mate, have given birth to three
cute dalmatian puppies! Little Dipper, Domino and Oddball...<br /><br />Starri
ng Eric Idle as Waddlesworth (the hilarious macaw), Glenn Close as Cruella her
self and Gerard Depardieu as Le Pelt (another baddie, the name should give a c
lue), this is a good family film with excitement and lots more!! One downfall
 of this film is that is has a lot of painful slapstick, but not quite as exce
ssive as the last film. This is also funnier than the last film.<br /><br />En
joy "102 Dalmatians"! :-)
'''
classify(text, inqtabs_dict, swn_dict)
```

Out[27]:
```
1
```

TextBlob

In [9]:
```
from textblob import TextBlob
```

In [10]:
```
# analysis = TextBlob(" ".join(words))

#Put the code under the classify method here.  Here's some code to start you o
ff
def classify(text, inqtabs_dict, swn_dict):

    text = re.sub('\[[^]]*\]', '', text)
    text = re.sub(r'\([^)]*\)', '', text)
    regex = r'[A-Za-z0-9]+' #You'll probably want to update this regular expre
ssion
    words = re.findall(regex, text)

    words = [w for w in words if not w in stop_words]

    #postagging = []


    #postagging.append(nltk.pos_tag(word_tokenize(" ".join(words))))


    #cleaned_text = ""



    #for word,y in postagging[0]:

        #wn_tag = penn_to_wn(y)

        #if wn_tag is None:

            #continue
        #else:

            #cleaned_text += word + " "

    sntmnt = TextBlob(" ".join(words))

    if sntmnt.sentiment.polarity > 0:
        score = 1
    else:
        score = 0

    return score
```

In [49]:
```
get_training_accuracy(data, inqtabs_dict, swn_dict)
```

```
Accuracy: 0.70044
```

Out[49]:
```
0.70044
```

Flair

In [14]:
```
from flair.models import TextClassifier
from flair.data import Sentence
classifier = TextClassifier.load('en-sentiment')
```

```
2022-05-08 19:31:29,616 loading file C:\Users\hp\.flair\models\sentiment-en-mix-distillbert_4.pt
```

In [16]:
```
def classify(text, inqtabs_dict, swn_dict):

    text = re.sub('\[[^]]*\]', '', text)
    text = re.sub(r'\([^)]*\)', '', text)
    regex = r'[A-Za-z0-9]+' #You'll probably want to update this regular expre
ssion
    words = re.findall(regex, text)

    words = [w for w in words if not w in stop_words]


    postagging = []


    postagging.append(nltk.pos_tag(word_tokenize(" ".join(words))))


    cleaned_text = ""

    for word,y in postagging[0]:

        wn_tag = penn_to_wn(y)

        if wn_tag is None:

            continue
        else:

            cleaned_text += word + " "
    #print(" ".join(words))
    #print(cleaned_text)

    sentence = Sentence(cleaned_text)

    classifier.predict(sentence)

    value = sentence.labels[0].to_dict()['value']

    if value == 'POSITIVE':
        result = sentence.to_dict()['labels'][0]['confidence']
    else:
        result = -(sentence.to_dict()['labels'][0]['confidence'])


    #print(sentence.to_dict()['labels'][0]['confidence'])

    if result > 0:
        score = 1
    else:
        score = 0

    #sntmnt = analyzer.polarity_scores(" ".join(words))['compound']

    #print(score)

    return score
```

In [ ]:
```
get_training_accuracy(data, inqtabs_dict, swn_dict)
```

In [22]:
```
get_training_accuracy(data, inqtabs_dict, swn_dict)
```

```
Accuracy: 0.82052
```

Out[22]:
```
0.82052
```