

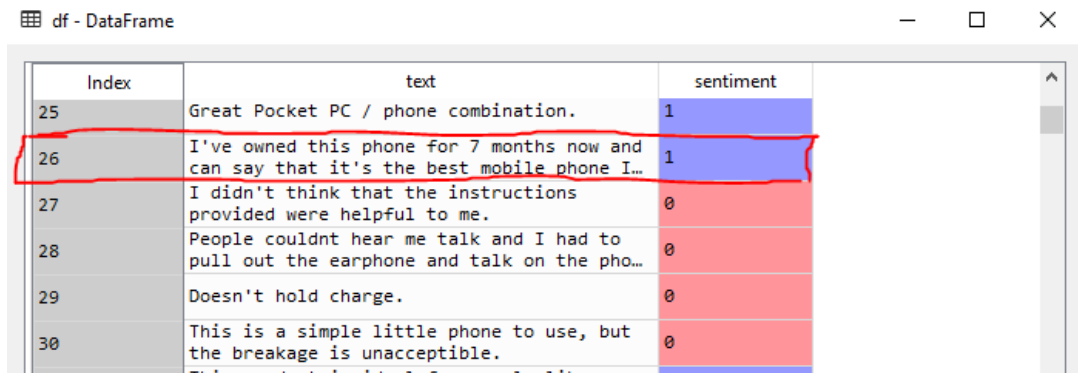
## Lab Session – Text Mining #1

**Data:** Please download the data set titled “amazon\_labelled.txt” to use with this lab session. This data set includes 1,000 product reviews randomly selected from Amazon.com. The file is tab-delimited. The 1<sup>st</sup> column contains the review text. The 2<sup>nd</sup> column indicates the manually labelled sentiment scores of reviews. The sentiment score is 1, if the review is labelled as positive and 0, if it is negative.

**Objective:** You need to complete 3 separate tasks for this lab: 1) manual pre-processing of data, 2) lexicon based sentiment analysis (i.e., sentiment analysis via a classification model), and 3) dictionary based sentiment analysis.

### Task 1: Manual Pre-processing of Text

- 1.1. Import the data set using Pandas’ read\_table method. Notice that you will need to name the columns yourself (e.g., 1<sup>st</sup> column is ‘text’ and 2<sup>nd</sup> column is ‘sentiment’).
- 1.2. Store the “text” of the review that is indexed at 26 into a new variable called “s”. Print this text.



Index	text	sentiment
25	Great Pocket PC / phone combination.	1
26	I've owned this phone for 7 months now and can say that it's the best mobile phone I...	1
27	I didn't think that the instructions provided were helpful to me.	0
28	People couldnt hear me talk and I had to pull out the earphone and talk on the pho...	0
29	Doesn't hold charge.	0
30	This is a simple little phone to use, but the breakage is unacceptable.	0

**Note:** For the following sub-tasks, you can use Python’s nltk library and list comprehensions.

- 1.3. Use the “word\_tokenize” function in nltk library to split the “s” variable into individual words (i.e., tokens). Compare the output to the output of the Python’s own split() method. Save the output from the “word\_tokenize” to a variable called “sTokenized”.

**Hint:** To import the “word\_tokenize” function, use the following line: “from nltk.tokenize import word\_tokenize”

- 1.4. Notice that some of the elements in the sTokenized list are contraction suffices (e.g., “’ve” and “’s”). Write a function to convert “’ve” to “have” and “’s” to “is”. Then, use this function to replace contracted words in the sTokenized with their full forms.

- 1.5. Two other issues in sTokenized are punctuations and numbers as tokens. Remove these types of elements from sTokenized.

**Hint:** Consider using a list comprehension with the isalpha() method. This method checks whether all characters in a string are alphabetic or not.

- 1.6. Convert all characters in all words in sTokenized to lowercase.

**Hint:** Consider using a list comprehension with the lower() method.

1.7. Use stemming to bring all the words in sTokenized to their root forms. In addition, remove the common English stop words from sTokenized.

**Hint:** Libraries / packages to use:

```
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
```

**QUESTION 1:** After these operations, how many tokens are remaining in sTokenized? Print out the final form of sTokenized.

## Task 2: Lexicon-based Sentiment Analysis (using Classification)

**Note:** The goal of this task is to build a classification model that can predict the sentiment score of a review using tokens as inputs. To do this, you will need to: 1) create a term-document matrix, 2) build and evaluate a classification model (using Decision Tree and SVM algorithms). Both of these operations can be performed using scikit-learn packages.

2.1. Use the CountVectorizer package (from sklearn.feature\_extraction.text import CountVectorizer) in scikit-learn to create a term-document matrix from the review texts. The output should look like the one below. Notice that you need to complete text pre-processing steps (i.e., cleaning, stop word removal, stemming) before creating the final term-document matrix.

**Hints:** “analyzer” parameter in CountVectorizer can be configured to read a custom function to do all the pre-processing operations in a single pass. You can first instantiate a custom analyzer using: analyzer = CountVectorizer().build\_analyzer() and then write a function that returns a list comprehension as: [stemmer.stem(word) for word in analyzer(doc) if word not in stopWords and word.isalpha()] to do this.

dfX - DataFrame

Index	abhor	abil	abl	abound	absolut	absolutel	ac	accept	access
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0

2.2. Once you have the term-document matrix, use it as an input to predict the manually labelled sentiment scores (1 or 0) in the original data. Use DT and SVM classification algorithms for model building. Use the holdout method with 70% training 30% testing allocation for evaluation purposes. Obtain the accuracy, precision, and recall of both DT and SVM algorithms.

**QUESTION 2:** Provide the accuracy, precision, and recall scores of DT and SVM models from the test data.

### Task 3: Dictionary-based Sentiment Analysis (using the VADER dictionary)



**Note:** The goal of this task is to identify the sentiments in the review texts using a dictionary based approach. The VADER (Valence Aware Dictionary and sEntimentReasoner) dictionary in the nltk library can be used for this purpose. For dictionary based approaches, the main idea is to check the sentiment score of each word in a given text from the dictionary and aggregate the scores to find the overall sentiment score of the text. Notice that you can simply use the original dataset for this purpose – i.e., there is no need for pre-processing or term-document conversion of the text.

3.1. Import the VADER dictionary and instantiate its object as follows:

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer  
sid = SentimentIntensityAnalyzer()
```

3.2. Write two functions: one that returns the positive sentiment score after applying “sid.polarity\_scores()” to a text document and another one that returns the negative sentiment score after applying “sid.polarity\_scores()” to a text document. Apply these functions on the “text” column in the original dataset to obtain the positive and negative sentiment scores.

3.3. Write a new function that compares the positive and negative sentiment scores of each text. If positive > negative, return 1, else return 0. Apply this function on the data to obtain a binary positive vs. negative sentiment indicator.

**QUESTION 3:** Compare the dictionary based sentiment indicator scores to the manually labelled sentiment scores in the original data. What percentage of sentiment scores match? Notice that this is essentially the accuracy score of the dictionary based sentiment analysis approach.