a1-sentiment-analysis-text-classification-dan-jang

October 10, 2023

1 CS410: Natural Language Processing, Fall 2023

1.1 A1: Sentiment Analysis Text Classification, Dan Jang - 10/9/2023

Description of Assignment

Introduction Our first assignment, A1: Sentiment Analysis Text Classification, will focus on processing the training & testing datasets of product-review data, then implementing a text-classification model based on two (2) different, suitable algorithms to process & predict the likelihood of 'negative' or 'positive' reviews, based on the title of each review.

Data Preparation A dataset containing product customer reviews, which is named the "Multilingual Amazon Reviews Corpus", in a json container format, with several columns. The assignment will focus on a smaller subset of the original dataset, where we will focus on **two (2) columns**: * "review_title" - self-explanatory * "stars" - an integer, either 1 or 5, where the former indicates "negative" and 5 indicates "positive."

There will be a training set & a test set.

We will load the dataset using Python & use respective libraries to implement our text-classification model.

Optionally, we will preprocess the data if needed, e.g. case-formating. ##### Feature Engineering We will choose a set of classifiers to focus on in our text-classification model, e.g. *n*-grams, num words, cue words, repeated punctuation, etc.

Text Classification Model To build our text-classification model, we will **follow these steps**: * Any *two* chosen suitable algorithms for text classification. * Vectorization of the text data (conversion of text for numerical features). * Training of the text-classification model using the training dataset, "sentiment_train.json." * Evaluation of our text-classification model using the testing dataset, "sentiment_test.json."

Results & Analysis A detailed analysis of the model's performance by comparing the results from the output of our two algorithms, where we will **include the following**: * *F1-score* or other relevant metrics. * Confusion matrix. * Any challenges or limitations of the text-classification model/task. * Suggestions for improvement in the performance of the text-classification model.

Requirements

1.1.1 Main Implementation: Text Classification

[1]:
```python
##### CS410: Natural Language Processing, Fall 2023 - 10/9/2023
##### A1: Sentiment Analysis Text Classification, Dan Jang
#### Objective: Exploring Natural Language Processing (NLP), by building a␣
↪text-classifier
#### for a text classification task, predicting whether a piece of text is␣
↪"positive" or "negative."

### 0.) Libraries
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import json
import pandas
import numpy as np
import matplotlib.pyplot as plot

# ### 1.) Main Program Wrapper, a1_text_classifer
#class a1_text_classifer(object):

### 1.2.a) Gaussian Näive Bayes algorithm using sklearn.naive_bayes.GaussianNB
### https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.
↪GaussianNB.html
### Returns four (4) thingys:
# I.) accuracy_score,
# II.) f1_score,
# III.) confusion_matrix,
# & IV.) classification_report.
def algo_one(xtrain, ytrain, xtest, ytest):
    gbayes = GaussianNB()

    gbayes.fit(xtrain, ytrain)
    predictionresults = gbayes.predict(xtest)

    return accuracy_score(ytest, predictionresults), f1_score(ytest,␣
↪predictionresults), confusion_matrix(ytest, predictionresults),␣
↪classification_report(ytest, predictionresults)
```

```python
### 1.2.b) Logistic Regression algorithm using sklearn.linear_model.
↪LogisticRegression
### https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.
↪LogisticRegression.html
### Returns four (4) thingys:
# I.) accuracy_score,
# II.) f1_score,
# III.) confusion_matrix,
# & IV.) classification_report.
def algo_two(xtrain, ytrain, xtest, ytest):
    lreg = LogisticRegression()

    lreg.fit(xtrain, ytrain)
    predictionresults = lreg.predict(xtest)

    return accuracy_score(ytest, predictionresults), f1_score(ytest,␣
↪predictionresults), confusion_matrix(ytest, predictionresults),␣
↪classification_report(ytest, predictionresults)

### 1.3.) NLTK Vader Lexicon-based Sentiment Analysis Classifier
### https://www.nltk.org/_modules/nltk/sentiment/vader.html
## sith_holocron = le sentiment intensity analyzer from NLTK
## theforce = le sentiment score
## aura = le text that is to be analyzed by [darth]vader from NLTK
def darth(aura):
    sith_holocron = SentimentIntensityAnalyzer()
    theforce = sith_holocron.polarity_scores(aura)
    # Debug Statement #2
    #print(theforce['compound'])
    return theforce['compound']

def main(): #trainfile, testfile):
    print("Welcome, this is the main program for A1: Sentiment Analysis Text␣
↪Classification.")
    print("Written by Dan J. for CS410: Natural Language Processing, Fall 2023.
↪")
    print("\nWe will use two classification algorithms:\n1. Gaussian Näive␣
↪Bayes\n& 2. Logistic Regression,\n...to create a text-classifier to guess␣
↪negative or positive sentimentiality based on various text-reviews of␣
↪products.")

    ## https://www.nltk.org/_modules/nltk/sentiment/vader.html
    print("Setting up & downloading the NLTK Vader sentiment analysis␣
↪classifier & lexicon...")
    nltk.download('vader_lexicon')
    print("...Vader has arrived.")
```

```python
## For converting accuracy to percent
percentness = float(100)

## 1.0.) Constants, Variables, & Datasets

# trainfile = str(trainfile)
# testfile = str(testfile)
traindata = []
testdata = []

# 1.0.I.A) Debug Statements #1a for dataset loading times:
print("Loading the training & testing datasets...")
# with open(trainfile, "r") as trainfile:
with open("sentiment_train.json", "r") as trainfile:
    #traindata = json.load(trainfile)
    for row in trainfile:
        traindata.append(json.loads(row))

trainframe = pandas.DataFrame(traindata)

# with open(testfile, "r") as testfile:
with open("sentiment_test.json", "r") as testfile:
    #testdata = json.load(testfile)
    for row in testfile:
        testdata.append(json.loads(row))

testframe = pandas.DataFrame(testdata)

# 1.0.I.B) Debug Statements #1b for dataset loading times:
print("Successfully loaded the training & testing datasets!\n")

## 1.0.1.) Initial Preprocessing of the training & testing data
## First, we isolate our two (2) columns, "review_title" & "stars."
## Second, we will convert values in the "stars" column so that 1␣
↪[negative] = 0 & 5 [positive] = 1.
## This will allow us to make the negative or positive sentiment a binary␣
↪value-based thingy.
trainframe = trainframe[['review_title', 'stars']]
trainframe['stars'] = trainframe['stars'].apply(lambda x: 1 if x == 5 else␣
↪0)

testframe = testframe[['review_title', 'stars']]
testframe['stars'] = testframe['stars'].apply(lambda x: 1 if x == 5 else 0)

## 1.0.1.) Applying NLTK Vader Sentiment
## https://www.nltk.org/_modules/nltk/sentiment/vader.html
```

```
    x2train = trainframe
    x2train = x2train[['review_title', 'stars']]
    # Have to truncate the training dataset so that it does not crash my␣
↪computer, heh.
    # Using a random_state seed of 2005, which was when Star Wars III was␣
↪released (when Vader was technically introduced in the prequelz).
    #x2train = x2train.sample(n=20000, random_state=2005)
    print("Now applying NLTK Vader sentiment analysis to the training dataset...
↪")
    x2train['nltk_vader_sentiment'] = x2train['review_title'].apply(darth)
    print("...Vader has been applied to the training set.")
    y2train = x2train['stars']

    x2test = testframe
    x2test = x2test[['review_title', 'stars']]
    print("Now applying NLTK Vader sentiment analysis to the testing dataset...
↪")
    x2test['nltk_vader_sentiment'] = x2test['review_title'].apply(darth)
    print("...Vader has been applied to the testing set.")

    ## 1.1.) Vectorization of the text-reviews in the datasets using sklearn.
↪feature_extraction.text.CountVectorizer.
    ## As a core component of text-classification, the vectorization process of␣
↪the text-review data is essential for feature engineering in natural␣
↪language processing.
    ## https://scikit-learn.org/stable/modules/generated/sklearn.
↪feature_extraction.text.CountVectorizer.html
    vectorization_machine_9000 = CountVectorizer()
    xtrain = vectorization_machine_9000.
↪fit_transform(trainframe['review_title'])
    xtrain = xtrain.toarray()
    ytrain = trainframe['stars']

    xtest = vectorization_machine_9000.transform(testframe['review_title'])
    xtest = xtest.toarray()
    ytest = testframe['stars']

    ## 1.1.1.) Applying NLTK Vader Sentiment to vectorized data
    x2traintext = vectorization_machine_9000.
↪fit_transform(x2train['review_title'])
    x2trainsentiment = x2train['nltk_vader_sentiment'].values.reshape(-1,1)
    parsed_x2traintext = pandas.DataFrame(x2traintext.toarray())
    parsed_x2trainsentiment = pandas.DataFrame(x2trainsentiment)

    x2testtext = vectorization_machine_9000.transform(x2test['review_title'])
    x2testsentiment = x2test['nltk_vader_sentiment'].values.reshape(-1,1)
```

```
    parsed_x2testtext = pandas.DataFrame(x2testtext.toarray())
    parsed_x2testsentiment = pandas.DataFrame(x2testsentiment)

    x2train = pandas.concat([parsed_x2traintext, parsed_x2trainsentiment],␣
↪axis=1)
    x2test = pandas.concat([parsed_x2testtext, parsed_x2testsentiment], axis=1)

    ### 1.0.2a) Run Algorithms & Print the Model Results - without classifiers␣
↪(with vectorization)
    print("-----\n")
    print("Running algorithms on le training & testing datasets (without␣
↪classifiers)...")
    print("Running Gaussian Näive Bayes algorithm, version A...")
    algo1accuracy, algo1f1, algo1cmatrix, algo1creport = algo_one(xtrain,␣
↪ytrain, xtest, ytest)
    print("..First algorithm is done!")

    print("Running Logistic Regression algorithm, version A...")
    algo2accuracy, algo2f1, algo2cmatrix, algo2creport = algo_two(xtrain,␣
↪ytrain, xtest, ytest)
    print("..Second algorithm is done!")

    print("...All Done!")
    print("-----\n")

    print("Here are le results [Version A ('control'), non-classification]...
↪\n")
    print("Algorithm #1, Version A: Gaussian Näive Bayes Performance, Metrics,␣
↪& Results:")
    print("...Accuracy was found to be, ", algo1accuracy * percentness, "%,")
    print("...F1 Score was found to be: ", algo1f1, ",")
    print("...with a Confusion Matrix: \n", algo1cmatrix, ",")
    print("...& lastly, the classification Report: \n", algo1creport)
    print("-----\n")

    print("Algorithm #2, Version A: Logistic Regression Performance, Metrics, &␣
↪Results:")
    print("...Accuracy was found to be, ", algo2accuracy * percentness, "%,")
    print("...F1 Score was found to be: ", algo2f1, ",")
    print("...with a Confusion Matrix: \n", algo2cmatrix, ",")
    print("...& lastly, the classification Report: \n", algo2creport)
    print("-----\n")

    ### 1.0.2b) Run Text-Classification Algorithms & Print the Model Results -␣
↪with NLTK Vader sentiment analysis (& vectorization)
    print("-----\n")
```

```
    print("Running algorithms on le training & testing datasets (with NLTK␣
↪Vader classifier)...")

    print("Running Gaussian Näive Bayes algorithm, version B...")
    algo1accuracy, algo1f1, algo1cmatrix, algo1creport = algo_one(x2train,␣
↪y2train, x2test, ytest)
    print("..First algorithm is done!")

    print("Running Logistic Regression algorithm, version B...")
    algo2accuracy, algo2f1, algo2cmatrix, algo2creport = algo_two(x2train,␣
↪y2train, x2test, ytest)
    print("..Second algorithm is done!")

    print("...All Done!")
    print("-----\n")

    print("Here are le results [Version B, NLTK Vader sentiment analysis␣
↪classification]...\n")
    print("Algorithm #1, Version B: Gaussian Näive Bayes Performance, Metrics,␣
↪& Results:")
    print("...Accuracy was found to be, ", algo1accuracy * percentness, "%,")
    print("...F1 Score was found to be: ", algo1f1, ",")
    print("...with a Confusion Matrix: \n", algo1cmatrix, ",")
    print("...& lastly, the classification Report: \n", algo1creport)
    print("-----\n")

    print("Algorithm #2, Version B: Logistic Regression Performance, Metrics, &␣
↪Results:")
    print("...Accuracy was found to be, ", algo2accuracy * percentness, "%,")
    print("...F1 Score was found to be: ", algo2f1, ",")
    print("...with a Confusion Matrix: \n", algo2cmatrix, ",")
    print("...& lastly, the classification Report: \n", algo2creport)
    print("-----\n")

#a1_program = a1_text_classifer("sentiment_train.json", "sentiment_test.json")

#### Commented out codez
# def main():

if __name__ == "__main__":
    main()
```

```
Welcome, this is the main program for A1: Sentiment Analysis Text
Classification.
Written by Dan J. for CS410: Natural Language Processing, Fall 2023.

We will use two classification algorithms:
```

```
1. Gaussian Näive Bayes
& 2. Logistic Regression,
…to create a text-classifier to guess negative or positive sentimentiality
based on various text-reviews of products.
Setting up & downloading the NLTK Vader sentiment analysis classifier &
lexicon…
…Vader has arrived.
Loading the training & testing datasets…

[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\Dan\AppData\Roaming\nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!

Successfully loaded the training & testing datasets!

Now applying NLTK Vader sentiment analysis to the training dataset…
…Vader has been applied to the training set.
Now applying NLTK Vader sentiment analysis to the testing dataset…
…Vader has been applied to the testing set.
-----

Running algorithms on le training & testing datasets (without classifiers)…
Running Gaussian Näive Bayes algorithm, version A…
..First algorithm is done!
Running Logistic Regression algorithm, version A…

c:\tools\miniconda3\lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

..Second algorithm is done!
…All Done!
-----

Here are le results [Version A ('control'), non-classification]…

Algorithm #1, Version A: Gaussian Näive Bayes Performance, Metrics, & Results:
…Accuracy was found to be,  59.199999999999996 %,
…F1 Score was found to be:  0.3664596273291925 ,
…with a Confusion Matrix:
 [[948  52]
 [764 236]] ,
…& lastly, the classification Report:
```

```
              precision    recall  f1-score   support

           0       0.55      0.95      0.70      1000
           1       0.82      0.24      0.37      1000

    accuracy                           0.59      2000
   macro avg       0.69      0.59      0.53      2000
weighted avg       0.69      0.59      0.53      2000

-----

Algorithm #2, Version A: Logistic Regression Performance, Metrics, & Results:
…Accuracy was found to be,  92.7 %,
…F1 Score was found to be:  0.9272908366533865 ,
…with a Confusion Matrix:
 [[923  77]
 [ 69 931]] ,
…& lastly, the classification Report:
              precision    recall  f1-score   support

           0       0.93      0.92      0.93      1000
           1       0.92      0.93      0.93      1000

    accuracy                           0.93      2000
   macro avg       0.93      0.93      0.93      2000
weighted avg       0.93      0.93      0.93      2000

-----

-----

Running algorithms on le training & testing datasets (with NLTK Vader
classifier)…
Running Gaussian Näive Bayes algorithm, version B…
..First algorithm is done!
Running Logistic Regression algorithm, version B…
```

```
c:\tools\miniconda3\lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
..Second algorithm is done!
```

```
…All Done!
-----

Here are le results [Version B, NLTK Vader sentiment analysis classification]…

Algorithm #1, Version B: Gaussian Näive Bayes Performance, Metrics, & Results:
…Accuracy was found to be,  59.3 %,
…F1 Score was found to be:  0.36899224806201547 ,
…with a Confusion Matrix:
 [[948  52]
 [762 238]] ,
…& lastly, the classification Report:
              precision    recall  f1-score   support

           0       0.55      0.95      0.70      1000
           1       0.82      0.24      0.37      1000

    accuracy                           0.59      2000
   macro avg       0.69      0.59      0.53      2000
weighted avg       0.69      0.59      0.53      2000

-----

Algorithm #2, Version B: Logistic Regression Performance, Metrics, & Results:
…Accuracy was found to be,  92.80000000000001 %,
…F1 Score was found to be:  0.9281437125748503 ,
…with a Confusion Matrix:
 [[926  74]
 [ 70 930]] ,
…& lastly, the classification Report:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93      1000
           1       0.93      0.93      0.93      1000

    accuracy                           0.93      2000
   macro avg       0.93      0.93      0.93      2000
weighted avg       0.93      0.93      0.93      2000

-----
```

1.1.2 Text-Classification Model Performance Analysis & Discussion

Initial Data Results, Metrics, & Analysis For Version 'A', which served as our vectorized, but non-classified (our 'control' run) iteration, where it ran the Gaussian Näive Bayes & Logistic Regression algorithms on the training & testing datasets - the following results were found at the initial successful attempt:

Gaussian Näive Bayes Results (Version A):

```
Accuracy: ~59.2%
F1 Score: ~0.3665

Confusion Matrix:
    [[948  52]
    [764 236]]

Classification Report:
            precision    recall  f1-score   support

        0       0.55      0.95      0.70      1000
        1       0.82      0.24      0.37      1000

accuracy                          0.59      2000
macro avg       0.69      0.59      0.53      2000
weighted avg    0.69      0.59      0.53      2000
```

Logistic Regression Results (Version A):

```
Accuracy: ~92.7%
F1 Score: ~0.92729

Confusion Matrix:
    [[923  77]
    [ 69 931]]

Classification Report:
            precision    recall  f1-score   support

        0       0.93      0.92      0.93      1000
        1       0.92      0.93      0.93      1000

accuracy                          0.93      2000
macro avg       0.93      0.93      0.93      2000
weighted avg    0.93      0.93      0.93      2000
```

For Version 'B', like the previous version, both algorithms were ran, but with the NLTK Vader Sentiment Analysis classifier being applied to the training & testing datasets to possibly improve text-classification:

Gaussian Näive Bayes Results (Version B):

```
Accuracy: ~59.3%
F1 Score: ~0.36899

Confusion Matrix:
    [[948  52]
    [764 238]]
```

```
Classification Report:
            precision    recall  f1-score   support

        0       0.55      0.95      0.70      1000
        1       0.82      0.24      0.37      1000

accuracy                            0.59      2000
macro avg       0.69      0.59      0.53      2000
weighted avg    0.69      0.59      0.53      2000
```

Logistic Regression Results (Version B):

```
Accuracy: ~92.7%
F1 Score: ~0.92729

Confusion Matrix:
    [[923  77]
    [ 69 931]]

Classification Report:
            precision    recall  f1-score   support

        0       0.93      0.93      0.93      1000
        1       0.93      0.93      0.93      1000

accuracy                            0.93      2000
macro avg       0.93      0.93      0.93      2000
weighted avg    0.93      0.93      0.93      2000
```

Comparative Analysis & Discussion It would appear that in Version A, where no classifiers were applied (ergo, serving as our 'control'), the Gaussian Näive Bayes algorithm was accurate only ~56% of the time, which is only ~6% better than the supposed 50-50 chance of guessing between the two, *binary* possibilities, of that being either a "negative" or "positive" review - where the former is represented by a 0 & latter represented by a 1.

Contrarywise, the Logistic Regression algorithm of Version A was accurate ~96% of the time, which showcases a highly significant increase in both relative (in comparsion to Gaussian Näive Bayes's ~56% accuracy) & absolute accuracy.

In Version B, where NLTK Vader Sentiment Analysis was applied as a classifer to both the training & testing datasets, we see that there was a very small, modest increase in accuracy in both algorithms (as seen above for Version B results), …where we see an increase of ~0.11% in accuracy for the Gaussian Näive Bayes algorithm …& an increase of ~0.01% in accuracy for Logistic Regression.

Interestingly, when NLTK Vader was ran during the debugging code-state where the NLTK Vader applied training dataset was limited to only 20k rows (vs. the full 80k rows), I observed a ~40% increase in accuracy for the Gaussian Näive Bayes algorithm to around ~97%. However, this might have been a misplaced variable naming issue, thus a fluke, as I had to fix some of the result-print statements outputting for the other algorithm, e.g. the Logistic Regression algorithm result-variables outputting for the Gaussian Näive Bayes algorithm, & vice-versa.

Text-Classification Challenges & Limitations It would appear that implementing both the Gaussian Näive Bayes & the Logistic Regression algorithms were straightforward as we were able to use sklearn.naive_bayes & sklearn.linear_model to use these algorithms in our text-classification model & program.

One small challenge was the time that NLTK Vader took to process the sentiments, particularly, for the classification of the training dataset, as it has 80k rows, ergo, 80k review_title values to process, which took a good amount of time to process.

However, after NLTK Vader was applied & then the algorithms were beginning to process the data, Version B training dataset had to be truncated in the number of rows from 80k to 20k, as my computer was running out & did eventually run out of allocatable RAM (~30 GB+ at the attempt), which shows a possible computational resource-based limitation to using NLTK Vader for sentiment analysis.

This made it slightly difficult to run the algorithm, ergo, the program in entirety to figure out bugs. However, this was fixable by initially debugging (then fixing to the smaller number of rows for Version B) with a smaller section of the data, e.g. 20000 rows, to verify that the Vader classification was actually working at first, before reverting back to 80000 rows.

Eventually, the full 80k rows of training data was able to be processed with the Vader classification, by closing RAM-intensive programs open at the time & restarting the Kernel to clear the cached RAM usages (~25 GB+ peak during eventual, successful full attempt).

Discussion for Future Performance & Efficacy Improvements The preprocessing, I do admit, may have been lacking. Besides the memory efficiency by truncating the .json files by only using the two (2) columns, “review_title” & “stars”, I think there could have been better preprocessing to improve performance, e.g. removing punctuation, delimitters, or etc. This was a pertinent thought for improvement whilst awaiting the completion of the NLTK Vader sentiment analysis applying to the training & testing data sets.

It also appears that Gaussian Näive Bayes, ran without classifiers or other changes (besides vectorization), performed ~40% weaker than its counterpart, the Logistic Regression algorithm. This might indicate a possible, inherent weakness of using Gaussian Näive Bayes for this specific facet of text-classification, thus, the exploration/usage of alternative, more efficient classification-algorithms may also be a possible method to improve both predictive accuracy & performance.

One whimsical thought of mine I had, was to possibly implement a custom classifer that places a high likelihood of review-negativity on the presence of common curse-words in the “review_title” & high likelihood of review-positivity on the presence of common ‘good-qualifying’ words, e.g. ‘great’, ‘awesome’, ‘amazing’, etc. However, this may take some time to tweak correctly & may be suspectible to cultural differences, grammatical quirks, suspectible to lexicon-shifts over time, needing to type out curse-words in a submitted assignment (which would be a little awkward, hehe), & other foreseeable hurdles if it were to be implemented - but would be both interesting & lead to possible performance & accuracy improvements.

1.1.3 References & Resources

Libraries & Dependencies

`matplotlib.pyplot`

numpy
pandas

sklearn.naive_bayes.GaussianNB

sklearn.linear_model.LogisticRegression

sklearn.model_selection.train_test_split

sklearn.feature_extraction.text.CountVectorizer

sklearn.metrics.f1_score

sklearn.metrics.accuracy_score

sklearn.metrics.confusion_matrix

sklearn.metrics.classification_report

nltk.sentiment.vader.SentimentIntensityAnalyzer

nbconvert

References & Credits *NLP Tutorial for Text Classification in Python* by Vijaya Rani

Using CountVectorizer to Extracting Features from Text by *GeeksforGeeks*

Special Thanks Fixing *sklearn ImportError: No module named _check_build*

Extra Stuff

1.1.4 Initial Full 80k-Row Processing Results Raw Output

```
Algorithm #1, Version A: Gaussian Näive Bayes Performance, Metrics, & Results:
...Accuracy was found to be,  59.199999999999996 %,
...F1 Score was found to be:  0.3664596273291925 ,
...with a Confusion Matrix:
[[948  52]
[764 236]] ,
...& lastly, the classification Report:
            precision    recall  f1-score   support

        0       0.55      0.95      0.70      1000
        1       0.82      0.24      0.37      1000

    accuracy                           0.59      2000
macro avg       0.69      0.59      0.53      2000
weighted avg       0.69      0.59      0.53      2000

-----

Algorithm #2, Version A: Logistic Regression Performance, Metrics, & Results:
...Accuracy was found to be,  92.7 %,
```

```
...F1 Score was found to be:  0.9272908366533865 ,
...with a Confusion Matrix:
[[923  77]
[ 69 931]] ,
...& lastly, the classification Report:
            precision    recall  f1-score   support

        0       0.93      0.92      0.93      1000
        1       0.92      0.93      0.93      1000

    accuracy                           0.93      2000
macro avg       0.93      0.93      0.93      2000
weighted avg       0.93      0.93      0.93      2000

-----

Algorithm #1, Version B: Gaussian Näive Bayes Performance, Metrics, & Results:
...Accuracy was found to be,  59.3 %,
...F1 Score was found to be:  0.36899224806201547 ,
...with a Confusion Matrix:
[[948  52]
[762 238]] ,
...& lastly, the classification Report:
            precision    recall  f1-score   support

        0       0.55      0.95      0.70      1000
        1       0.82      0.24      0.37      1000

    accuracy                           0.59      2000
macro avg       0.69      0.59      0.53      2000
weighted avg       0.69      0.59      0.53      2000

-----

Algorithm #2, Version B: Logistic Regression Performance, Metrics, & Results:
...Accuracy was found to be,  92.80000000000001 %,
...F1 Score was found to be:  0.9281437125748503 ,
...with a Confusion Matrix:
[[926  74]
[ 70 930]] ,
...& lastly, the classification Report:
            precision    recall  f1-score   support

        0       0.93      0.93      0.93      1000
        1       0.93      0.93      0.93      1000

    accuracy                           0.93      2000
macro avg       0.93      0.93      0.93      2000
```

```
weighted avg       0.93      0.93      0.93      2000

-----
```

[ ]:
```
##### Juypter Notebook -> PDF Conversion thingy

#!pip install nbconvert
!jupyter nbconvert --to pdf a1-sentiment-analysis-text-classification-dan-jang.
 ↪ipynb
```