Sentiment analysis from tweets using SVM

**Introduction:** Language serves as a structured medium through which human beings communicate, whether through spoken words or written text. Expressions like "Blah blah," "LOL," or "hmm..." may resonate with us, but the challenge lies in whether computers can interpret them. However, machines lack the ability to comprehend such expressions or any textual content, They only understand numbers.So, over the decades scientists have been researching how to make machines understand our language. And thus they developed all the Natural Language Processing or NLP Techniques.

**Important imports used** : !pip install nltk, import nltk, import regex as re

**Dataset used :** Provided in the file ‘sentiment-dataset.tsv’.

# Data input and pre-processing:

**parse\_data\_line()** function takes a tab-separated line (data\_line) and splits it into the label and statements (text content) assuming they are separated by tabs

**pre\_process()** function performs text processing steps to tokenize a given string of text into a list of tokens (words).

Regex used to remove punctuation at the ends or beginnings of words.

# Simple feature extraction:

The **to\_feature\_vector() function** primarily populates two dictionaries:

* global\_feature\_dict: A global dictionary maintaining all unique tokens in the dataset and updates this dictionary by adding tokens as keys.
* feature\_vector: A local dictionary representing the features of a specific text and It assigns 1 to each token which is present in the tokens list.

# Cross-validation on training data:

**cross\_validate()** function, performs a 10-fold cross-validation on the training data for sentiment analysis. Split the dataset into training and testing sets for each fold. Extract features and labels from the training and testing sets. Train the classifier and Predict labels for the test set using the trained classifier.

Calculate **performance metrics such as precision, recall, F1-score, and accuracy from sklearn.metrics import precision\_recall\_fscore\_support** while storing the computed metrics for each fold in the `results` list and then compute the average performance metrics across all folds, Store the aggregated metrics in the `cv\_results` dictionary and return it.

# Error analysis:

Improving preprocessing in sentiment analysis involves Removing Noise, Normalization,Tokenization, Stopwords Removal, Lemmatization and Stemming, punctuation removal. Further ,The Heatmap depicting False positive and False negative values in the Confusion matrix highlighted a bias towards positive outcomes. Analysis revealed a higher frequency of positive words, indicating disproportionate training on positive datasets and thus the model is biased towards positive values as the frequency of positive words appeared more, and thus training has been done on positive datasets more.

**NOTE: Sentiment\_FN\_FP.csv file** , stores the value of false positive and false negative value.

# Optimizing pre-processing and feature extraction:

1. **pre\_process() function**- In addition to punctuation removal above, I am performing and stop word removal, then I am tokenizing out text using nltk tokenizer further I am performing normalization and lemmatization as well, when I tried removing hyperlink explicitly the overall accuracy deceased , with 1 percent.
2. **to\_feature\_vector function** - I explored diverse methods, initially employing unigram and then progressing through Markov model n-grams to optimize text. Eventually, I opted for bigram with unigram analysis to refine the given text for enhanced accuracy.
3. **Classifier optimization** - involves adjusting weights for negative and positive sentiments, setting the regularization parameter (C) to 1.0, and extending iterations to 10,000. Though each modification individually had minor effects, collectively they raised the model's accuracy to 0.85.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Base | First Iteration | Final Iteration |
| **Evaluate** | {'precision': 0.8470426013416799,  'recall': 0.848318441342039,  'f1\_score': 0.8473022431744743,  'accuracy': 0.848318441342039} | {'precision': 0.850944093976398,  'recall': 0.8528994030416162,  'f1\_score': 0.8503121194506205,  'accuracy': 0.8528994030416162} | {'precision': 0.8527711341492573,  'recall': 0.8546509373751942,  'f1\_score': 0.8521183671878161,  'accuracy': 0.8546509373751942} |
| **Pre processing** | filtering start and end punctuation | Hyperlink , Normalisation, lemmatising , stop word, filtering start and end punctuation | Normalisation, lemmatising , stop word, filtering start and end punctuation |
| **Feature extraction** | Unigram | Bi-gram , Unigram | Bi-gram , Uniigram |
| **SVM**  **Parameter** | Default | dual='auto', C=1.0, class\_weight=class\_weigh t\_value, max\_iter= 100000 | dual=True, C=1.0, class\_weight=None, max\_iter= 1000 |

**Conclusion:**

In this coursework, after sentiment analysis using SVM, the model achieved a notable accuracy improvement from **0.847 to 0.854** after optimizing the pre- processing step with lemmatization, stop words, punctuation removal and unwanted tokens. Moreover, feature extraction also played an important role in optimization , using the combination of unigram and bigram helped decrease the biases of our classifier to some degree. The SVM model, LinearSVC, provides a lot of implicit parameters like cost parameter, class weight, penalty and loss for fine tuning our classification metrics.