NLP IST 664

Final Project -
Kaggle competition
movie review phrase data

Professor:
Michael Larche

Group:
Sahil Wani
Harshit Joshi

Dataset:

The Rotten Tomatoes dataset is divided into tab-separated files that include phrases. It includes movie reviews from the website Rotten Tomatoes, which total roughly 156,060 phrases and have the following overall sentiment:

The sentiment labels are:

0: negative

1: somewhat negative

2: neutral

3: somewhat positive

4: positive

Columns in dataset are:

- PhraseId : Unique id for phrase
- Sentence ID: unique identification for sentence
- Phrase: part of phrase/ entire phrase
- Sentiment: Integer value for type of sentiment as described above

Dataset SS:

```
PhraseId        SentenceId      Phrase  Sentiment
1       1       A series of escapades demonstrating the adage that what is good for
2       1       A series of escapades demonstrating the adage that what is good for
3       1       A series        2
4       1       A       2
5       1       series  2
6       1       of escapades demonstrating the adage that what is good for the goose
7       1       of      2
8       1       escapades demonstrating the adage that what is good for the goose
9       1       escapades       2
10      1       demonstrating the adage that what is good for the goose 2
11      1       demonstrating the adage 2
12      1       demonstrating   2
13      1       the adage       2
14      1       the     2
15      1       adage   2
```

Objective:

Goal is to perform Classification on the dataset and gauge machine learning models based on evaluation metrics namely accuracy precision recall and F1 score. This requires, to preprocess the dataset, create features, use different lexicons for sentiment analysis and then run models.

Additionally, we have also created combined existing features, cross-validation on our dataset and test it with Naïve Bayes (nltk), SVM and Random Forest for Sklearn and also cross validation of Random Forest before calculating model metrics.

Step 1: Preprocessing

Perform data preprocessing:

Starting off with running the python scripts in command prompt, our processkaggle() is only function called by main function. It is solely responsible to call all other functions that are important for preprocessing, evaluating metrics, and calling other files for running algorithms.

Parameters for processkaggle(): dirPath and LimitStr

```python
# our main function- processkaggle
def processkaggle(dirPath,limitStr):

  limit = int(limitStr)
  os.chdir(dirPath)
  f = open('./train.tsv', 'r')
  phrasedata = []

  for line in f:
    # ignore the first line starting with Phrase and read all lines
    if (not line.startswith('Phrase')):
      # remove final end of line character
      line = line.strip()
      # each line has 4 items separated by tabs
      # ignore the phrase and sentence ids, and keep the phrase and sentiment
      phrasedata.append(line.split('\t')[2:4])

  # pick a random sample of length limit because of phrase overlapping sequences
  random.shuffle(phrasedata)
  phraselist = phrasedata[:limit]
  print('Read', len(phrasedata), 'phrases, using', len(phraselist), 'random phrases')
```

a. Tokenize words

- For this we defined function called tokenize() which takes in phraselist (list of phrases) and tokenizes every word in a phrase
- Convert every tokenized word to lower case, and
- Remove stopwords from these words.

```python
#Tokenized the phrases, turned to lowercase and removed stop words
def tokenize(phraselist):
  phrasedocs = []

  for phrase in phraselist:
    #Tokenization of phrases
    phrase_tokens = nltk.word_tokenize(phrase[0])
    #turning every word to lower case
    phrase_tokens = [w.lower() for w in phrase_tokens]
    #removing stopwords. Here we added some of our own stopwords which were not included in nltk
    nltkstopwords = nltk.corpus.stopwords.words('english')
    morestopwords = ['could', 'would', 'might', 'must', 'need', 'sha', 'wo', 'y', "'s", "'d", "'ll", "'t", "'m", "'re", "'ve"]
    #Final stopwords list
    stopwords = nltkstopwords + morestopwords
    phrase_tokens = [word for word in phrase_tokens if not word in stopwords]
    phrasedocs.append((phrase_tokens, int(phrase[1])))
  return phrasedocs
```

b. Convert to lowercase
    - with help of tokenize() function, we were able to convert every tokened word in our phraselist to lower case.

c. Remove stop words
    - To remove stopwords, we used stopwords from the NLTK library for the English language.
    - Additionally we also added a few words on our own that we not included in the initial nltk stopwords list

d. Remove non alphabetic characters:
    - For this we defined a new function using Regular Expression library with regex.
    - Our aim was to remove non alphabetic words which might not have any importance in classifying sentiments like numbers

```
#removing non alphabetic characters
def alpha_filter(l):
    # filter out non-alphabetic words which don't have relevance

    for i in l:
      pattern = re.compile('^[^A-Za-z]+$')
      if pattern.match(i):
          continue
      return i
```

e. Reduce words to their stemmed version:
    - For stemming we defined new function called stemmer() which converts words to their base/ root form as in English Language.
    - For this function we choose PorterStemmer as it is widely accepted algorithm, efficient computation and because its primarily designed for English words.

```
#using the poter stemming for normalizing
def stemmer(phraselist):
    phrase_stemmed = []

    if phraselist is not None:
        for phrase in phraselist:
            if phrase is not None and phrase[0] is not None:
                porter = nltk.PorterStemmer()
                stemmed_words = [porter.stem(t) for t in phrase[0]]
                phrase_stemmed.extend([(stemmed_word, int(phrase[1])) for stemmed_word in stemmed_words])
    return phrase_stemmed
```

f. Regex_clean/ substitution of words:
    - For specific words like 'll , wo n't, that 's which have not been correctly tokenize and need further processing, we

created a new function regex_clean() which iterates through phrases and substitute's these words with correct words.

- For eg, word 'that 's' is converted to 'that is'. 're is converted to 'are'.

```
#substituting the short form of words with full form with the help of regex
def regex_clean(doc):
    regex = []

    for review_text in doc:
        review_text = re.sub(r"that 's","that is",review_text)
        review_text = re.sub(r"\bcannot\b", "can not", review_text)
        review_text = re.sub(r"\bain't\b", "am not", review_text)
        review_text = re.sub(r"\'ve","have",  review_text)
        review_text = re.sub(r"\bno\b", "not",review_text)
        review_text = re.sub(r"wo n't","will not", review_text)
        review_text = re.sub(r"do n't","do not", review_text)
        review_text = re.sub(r"\bdoesn't\b", "does not", review_text)
        review_text = re.sub(r"n\'t","not", review_text)
        review_text = re.sub(r"\'re","are",review_text)
        review_text = re.sub(r"\'d", "would", review_text)
        review_text = re.sub(r"\'ll", "will", review_text)
        review_text = re.sub(r"\'m", "am",review_text)
        review_text = re.sub(r"it 's","it is", review_text)
        regex.append(review_text)
    return regex
```

Step 2: Feature Creation

1) Bag of words:

   - For creation of feature sets, one of our approaches was to create bow ~ Bag of words function which returns most common words in our phrase (movie reviews)
   - This bow() function is used to create 2 different feature sets:

```
#BAG OF WORDS feature:
#getting most common words in phrases:
def bow(phrase, most_freq_words):
    phrase = nltk.FreqDist(phrase)
    features = [word for (word, count) in phrase.most_common(most_freq_words)]
    return features
```

1. One Filtered featured set: 'preprocessed_fs'

```
#filtered featured sets:
preprocessed_fs = [(unigram(token, processed_features), sentiment) for (token, sentiment) in phrasedocs]
```

2. One Unfiltered Feature set: 'tokenized_fs'

```
#unfiltered featured sets:
tokenized_fs = [(unigram(token, only_tokenized_features), sentiment) for (token, sentiment) in phrasedocs]
```

2) Unigram: For getting our feature sets in desired format for processing, we created unigram() function which takes in a word to be processed, and return true false based on whether it is in our processed list (output of above segments).

```
#Unigram features:
#This function generates dictionary of single word features
def unigram( phrase, word_features):
    phrase_words = set(phrase)
    features = {}
```

3) Sl_features: calculates sentiment for a given phrase, it uses phrases , processed_tokens and Sentiment_Lexicons as input. Sentiment_Lexicons has words that are labeled with type(weaksubj, strongsubj) and priorpolarity(positive, negative):
    a. Words that are weaksubj or strongsub but have positive priorpolarity are labelled as positive words
    b. Words that are weaksubj or strongsub but have positive priorpolarity are labelled as positive words

```
#The sl_features function generates a dictionary of sentiment-related features including the count of positive and negative sentiment-related words
def sl_features(phrase, processed_token, Sentiment_Lexicon):
  doc_words = set(phrase)
  features = {'positive':0, 'negative':0}

  for w in processed_token:
    features['V_{}'.format(w)] = (w in doc_words)

  for w in doc_words:
    if w in Sentiment_Lexicon:
      strength, posTag, isStemmed, polarity = Sentiment_Lexicon[w]
      if strength == 'weaksubj' and polarity == 'positive':
        features['positive'] += 1
      elif strength == 'strongsubj' and polarity == 'positive':
        features['positive'] += 2
      elif strength == 'weaksubj' and polarity == 'negative':
        features['negative'] += 1
      elif strength == 'strongsubj' and polarity == 'negative':
        features['negative'] += 2
  return features
```

4) Pos: This function calculates part of speech feature of a given phrase. First I t converts phrases to set of unique words in phrases. It also uses nltk.pos_tag() which uses POS tagging using NLTK library. It creates empty dictionary features and iterates over processed tokens. If the token is present or absent in unique words in phrases, creates a count for nouns, verb, adjectives, adverb. It keeps adding to these count based on if our token is noun, verb, adject or an adverb.

```python
#The pos function generates a dictionary of part-of-speech (POS)-related features in the phrase and the count of different POS categories (nouns, verbs, adjectives, and a
def pos(phrase, processed_token):
    unique_words = set(phrase)
    tagged_words = nltk.pos_tag(phrase)
    features = {}

    for word in processed_token:
        features['contains({})'.format(word)] = (word in unique_words)
    numNoun = 0
    numVerb = 0
    numAdj = 0
    numAdverb = 0

    for (word, tag) in tagged_words:
        if tag.startswith('N'): numNoun += 1
        if tag.startswith('V'): numVerb += 1
        if tag.startswith('J'): numAdj += 1
        if tag.startswith('R'): numAdverb += 1

    features['nouns'] = numNoun
    features['verbs'] = numVerb
    features['adjectives'] = numAdj
    features['adverbs'] = numAdverb
    return features
```

5) Liwc: it calculates LIWC (Linguistic Inquiry and Word Count) . Like POS tag, first it also converts phrases to unique words. Next, It will check if our token that is in our phrase_words. Likewise, it will check label of that word in poslist or neglist in Sentiment_Lexicon. Further, we had to import sentiment_read_LIWC_pos_neg_words script that contains isPresent function which checks if a particular word is present in poslist and neglist respectively. Based on this we counted number of positive and negative words as found by liwc function.

```python
#The liwc function generates a dictionary of LIWC (Linguistic Inquiry and Word Count) related features based on the given phrase as well as the count of positive and nega
def liwc(phrase, processed_token,poslist,neglist):
    phrase_words = set(phrase)
    features = {'positive':0, 'negative':0}

    for word in processed_token:
        features['contains({})'.format(word)] = (word in phrase_words)

    for word in phrase_words:
        if sentiment_read_LIWC_pos_neg_words.isPresent(word,poslist):
            features['positive']  += 1
        if sentiment_read_LIWC_pos_neg_words.isPresent(word,neglist):
            features['negative'] += 1
    return features
```

6) liwc_sl: A combination of liwc (Linguistic Inquiry and Word Count) and Sentiment Lexicon. Main goal of this function is to create a feature set which takes in a phrase to be classified, poslist, neglist, Sentiment_Lexicon and processed token. It then creates a unique list from phrases. Then it will check if our processed words are found in unique phrase list and adds 2 to positive/negative list if found with sentiment_read_LIWC_pos_neg_words.isPresent() function. Also it will add 1 to the lists if weaksubj strength found in Sentiment_Lexicon and 2 to the lists if strongsubj strength. This way it is combining functionality of both liwc and sl function.

```python
#The liwc_sl function generates a dictionary of LIWC (Linguistic Inquiry and Word Count) and sentiment-related features also it keep count of positive and negative words
#basically it is combination of sl_features() and liwc().
def liwc_sl(phrase, processed_token, Sentiment_Lexicon, poslist, neglist):
  phrase_words = set(phrase)
  features = {'positive':0, 'negative':0}

  for w in processed_token:
    features['contains({})'.format(w)] = (w in phrase_words)

  for w in phrase_words:
    if sentiment_read_LIWC_pos_neg_words.isPresent(w, poslist):
      features['positive'] += 2
    elif sentiment_read_LIWC_pos_neg_words.isPresent(w, neglist):
      features['negative'] += 2
    elif w in Sentiment_Lexicon:
      strength, posTag, isStemmed, polarity = Sentiment_Lexicon[w]
      if strength == 'weaksubj' and polarity == 'positive':
        features['positive'] += 1
      elif strength == 'strongsubj' and polarity == 'positive':
        features['positive'] += 2
      elif strength == 'weaksubj' and polarity == 'negative':
        features['negative'] += 1
      elif strength == 'strongsubj' and polarity == 'negative':
        features['negative'] += 2
  return features
```

Do you mind taking a quick feedback survey?

Additional functions and concepts used for running algorithms and calculating metrics:

1) Evaluation_metrics : This function was created to calculate average accuracy, precision, recall and F1 score after training naiveBayesclassifier, Sklearn SVM and Random Forest. It takes our data as trainset and test set and calculates the metrics with the classifier.

   a. Recall: In the context of sentiment analysis on the Rotten Tomatoes dataset, recall measures the ability of a classification model to correctly identify positive or negative sentiments among all the actual positive or negative sentiments in the dataset.

   b. Accuracy: Accuracy evaluates the overall correctness of a sentiment classification model by measuring the proportion of correctly predicted sentiments (positive or negative) out of the total sentiments in the Rotten Tomatoes dataset.

   c. Precision: Precision assesses the accuracy of a sentiment classification model in predicting positive or negative sentiments by measuring the proportion of correctly predicted positive or negative sentiments out of all the predicted positive or negative sentiments.

   d. F1 Score: The F1 score is a balanced measure that combines precision and recall in sentiment analysis on the Rotten Tomatoes dataset. It provides an overall assessment of the model's ability to predict positive and negative sentiments, considering both false positives and false negatives.

Therefore, we made a evaluation metrics function for Naïve bayes NLTK, SKlearn SVM and Random Forest:

NLTK Naïve Bayes:

```python
#This function trains a Naive Bayes classifier on the given training set and computes evaluation metrics (accuracy, precision, recall, and F1 score)
def evaluation_metrics(train_set, test_set):
    classifier = nltk.NaiveBayesClassifier.train(train_set)
    true_labels = []
    predicted_labels = []

    for features, label in test_set:
        predicted_label = classifier.classify(features)
        true_labels.append(label)
        predicted_labels.append(predicted_label)

    accuracy = nltk.classify.accuracy(classifier, test_set)
    precision = precision_score(true_labels, predicted_labels,average='weighted', zero_division=1)
    recall = recall_score(true_labels, predicted_labels, average='weighted', zero_division=1)
    F1 = f1_score(true_labels, predicted_labels, average='weighted', zero_division=1)
    return accuracy, precision, recall, F1
```

SKlearn SVM:

```python
#This function trains a SK learn  classifier on the given training set and computes evaluation metrics (accuracy, precision, recall, and F1 score)
def evaluation_metrics_SVM(train_set, test_set):
    classifier = SklearnClassifier(SVC(kernel='linear'))
    classifier.train(train_set)
    true_labels = []
    predicted_labels = []

    for features, label in test_set:
        predicted_label = classifier.classify(features)
        true_labels.append(label)
        predicted_labels.append(predicted_label)

    accuracy = nltk.classify.accuracy(classifier, test_set)
    precision = precision_score(true_labels, predicted_labels,average='weighted', zero_division=1)
    recall = recall_score(true_labels, predicted_labels, average='weighted', zero_division=1)
    F1 = f1_score(true_labels, predicted_labels, average='weighted', zero_division=1)
    return accuracy, precision, recall, F1
```

Sklearn Random Forest:

```python
#This function trains a Random Forest classifier on the given training set and computes evaluation metrics (accuracy, precision, recall, and F1 score)
def evaluation_metrics_RF(train_set, test_set):
    classifier = SklearnClassifier(RandomForestClassifier())
    classifier.train(train_set)
    true_labels = []
    predicted_labels = []

    for features, label in test_set:
        predicted_label = classifier.classify(features)
        true_labels.append(label)
        predicted_labels.append(predicted_label)

    accuracy = nltk.classify.accuracy(classifier, test_set)
    precision = precision_score(true_labels, predicted_labels,average='weighted', zero_division=1)
    recall = recall_score(true_labels, predicted_labels, average='weighted', zero_division=1)
    F1 = f1_score(true_labels, predicted_labels, average='weighted', zero_division=1)
    return accuracy, precision, recall, F1
```

2) Sklearn : To use SVM classifier and random forest classifier from sklearn altogether, we designed this function which will take in data, and percent partition between train and test set. It then runs the classifier for both

algorithms and calculates and prints evaluation metrics: accuracy, precision, recall and F1 score.

```
# using linear regression classifier and random forest clasifier from
sklearn to train our model and compare it with nltk naive bayes results
def sklearn(features,percent):
  training_size = int(percent*len(features))

  #Split the features into a training set and a test set
  train_set, test_set = features[training_size:], features[:training_size]

  #Train a Linear Regression classifier using the SklearnClassifier
  wrapper
  classifier1 = SklearnClassifier(SVC(kernel='linear'))
  classifier1.train(train_set)

  print("SVM Classification sklearn")
  accuracy, precision, recall, F1 = evaluation_metrics_SVM(train_set,
  test_set)
  print("Evaluation Metric: Accuracy={:.4f}, Precision={:.4f}, Recall={:.
  4f}, F1 Score={:.4f}".format(accuracy, precision, recall, F1))

  #Train a Random Forest classifier using the SklearnClassifier wrapper
  classifier2 = SklearnClassifier(RandomForestClassifier())
  classifier2.train(train_set)

  print("Random Forest sklearn")
  accuracy, precision, recall, F1 = evaluation_metrics_RF(train_set,
  test_set)
  print("Evaluation Metric: Accuracy={:.4f}, Precision={:.4f}, Recall={:.
  4f}, F1 Score={:.4f}".format(accuracy, precision, recall, F1))
```

3) random_forest_classification: To implement cross validation of random forest classifier, we created this function which takes in number of folds, feature sets, labels. This function calculates train and test sets for each fold. It also calculates evaluation metrics like accuracy, precision, recall, f1 score with eval_measures by comparing reflist (reference label) and testlist. Reason why we chose Random Forest for this task is because random forest is great is it provides high accuracy in classification tasks with multiple decision trees, handling high dimensional data (useful when we combine features)

```
# It perform 5 fold cross validation using random forest classification
def random_forest_classification(num_of_folds, feature_set, labels):
    subset_size = int(len(feature_set) / num_of_folds)
    accuracy_list = []
    reflist = []
    testlist = []
    print("Random Forests Classifier")

    for i in range(num_of_folds):
        print('Starting Fold', i)
        current_test = feature_set[i * subset_size:][:subset_size]
        current_train = feature_set[:i * subset_size] + feature_set[(i + 1) * subset_size:]

        # training our model
        classifier = SklearnClassifier(RandomForestClassifier())
        classifier.train(current_train)
        current_accuracy = nltk.classify.accuracy(classifier, current_test)
        print('fold-{}, Accuracy-{}'.format(i, current_accuracy))
        accuracy_list.append(current_accuracy)
        for (features, label) in current_test:
            reflist.append(label)
            testlist.append(classifier.classify(features))

    print('mean accuracy-', sum(accuracy_list) / num_of_folds)
    (precision_list, recall_list, F1_list) = eval_measures(reflist, testlist, labels)
    eval_metrics(precision_list, recall_list, labels)
```

4) Cross-Validation: Cross validation is used to evaluate performance of machine learning model on test data like our movie reviews. It is helpful as it provides more reliable performance evaluation without relying on train-test split.

Step 3:

1) NLTK Naïve Bayes Classification:

   Metrics for Filtered sets:

```
_______________________ Filtered Feature Set Naive Bayes Classification _______________________

Preprocessed_fs naive bayes:
Evaluation Metric: Accuracy=0.4833, Precision=0.7503, Recall=0.4833, F1 Score=0.3150

sl_preprocessed naive bayes:
Evaluation Metric: Accuracy=0.5200, Precision=0.4486, Recall=0.5200, F1 Score=0.4382

pos_preprocessed naive bayes:
Evaluation Metric: Accuracy=0.4967, Precision=0.4424, Recall=0.4967, F1 Score=0.4235

liwc_preprocessed naive bayes:
Evaluation Metric: Accuracy=0.5500, Precision=0.5482, Recall=0.5500, F1 Score=0.5171

 sl_liwc_preprocessed naive bayes:
Evaluation Metric: Accuracy=0.5433, Precision=0.4666, Recall=0.5433, F1 Score=0.4778
```

Unfiltered Set:

```
_____________________Unfiltered Feature Set Naive Bayes Classification______________________

 tokenized_fs naive bayes:
Evaluation Metric: Accuracy=0.4900, Precision=0.4415, Recall=0.4900, F1 Score=0.4048

 sl_tokenized naive bayes:
Evaluation Metric: Accuracy=0.4967, Precision=0.4056, Recall=0.4967, F1 Score=0.4150

 pos_tokenized naive bayes:
Evaluation Metric: Accuracy=0.5133, Precision=0.4379, Recall=0.5133, F1 Score=0.4399

 liwc_tokenized naive bayes:
Evaluation Metric: Accuracy=0.4867, Precision=0.4270, Recall=0.4867, F1 Score=0.4035

 sl_liwc_tokenized naive bayes:
Evaluation Metric: Accuracy=0.5100, Precision=0.4696, Recall=0.5100, F1 Score=0.4328
```

2) Sk Learn Classification:

Filtered Set with SVM Classification and Random Forest Classification:

```
______________________Filtered Feature Set SkLearn Classification______________________

preprocessed_fs Sklearn Evaluation Metrics :
The exact solution is  x = 0
SVM Classification sklearn
Evaluation Metric: Accuracy=0.4833, Precision=0.7503, Recall=0.4833, F1 Score=0.3150
Random Forest sklearn
Evaluation Metric: Accuracy=0.4833, Precision=0.7503, Recall=0.4833, F1 Score=0.3150

sl_preprocessed Sklearn Evaluation Metrics :
SVM Classification sklearn
Evaluation Metric: Accuracy=0.5200, Precision=0.4823, Recall=0.5200, F1 Score=0.4046
Random Forest sklearn
Evaluation Metric: Accuracy=0.5067, Precision=0.4363, Recall=0.5067, F1 Score=0.4575

pos_preprocessed Sklearn Evaluation Metrics :
SVM Classification sklearn
Evaluation Metric: Accuracy=0.4833, Precision=0.7503, Recall=0.4833, F1 Score=0.3150
Random Forest sklearn
Evaluation Metric: Accuracy=0.4667, Precision=0.3679, Recall=0.4667, F1 Score=0.3988

liwc_preprocessed Sklearn Evaluation Metrics :
SVM Classification sklearn
Evaluation Metric: Accuracy=0.5267, Precision=0.4821, Recall=0.5267, F1 Score=0.4429
Random Forest sklearn
Evaluation Metric: Accuracy=0.5200, Precision=0.5020, Recall=0.5200, F1 Score=0.4705

sl_liwc_preprocessed Sklearn Evaluation Metrics :
SVM Classification sklearn
Evaluation Metric: Accuracy=0.5267, Precision=0.6433, Recall=0.5267, F1 Score=0.4185
Random Forest sklearn
Evaluation Metric: Accuracy=0.5000, Precision=0.4247, Recall=0.5000, F1 Score=0.4156
```

Unfiltered Set with SVM Classification and Random Forest Classification:

```
_______________________Unfiltered Feature Set SkLearn Classification_______________________

tokenized_fs Sklearn Evaluation Metrics :
SVM Classification sklearn
Evaluation Metric: Accuracy=0.4867, Precision=0.4132, Recall=0.4867, F1 Score=0.4137
Random Forest sklearn
Evaluation Metric: Accuracy=0.4833, Precision=0.4738, Recall=0.4833, F1 Score=0.3399

sl_tokenized Sklearn Evaluation Metrics :
SVM Classification sklearn
Evaluation Metric: Accuracy=0.5267, Precision=0.4869, Recall=0.5267, F1 Score=0.4808
Random Forest sklearn
Evaluation Metric: Accuracy=0.5133, Precision=0.4802, Recall=0.5133, F1 Score=0.3989

 pos_tokenized Sklearn Evaluation Metrics :
SVM Classification sklearn
Evaluation Metric: Accuracy=0.4800, Precision=0.4175, Recall=0.4800, F1 Score=0.4301
Random Forest sklearn
Evaluation Metric: Accuracy=0.4900, Precision=0.4573, Recall=0.4900, F1 Score=0.3457

liwc_tokenized  Sklearn Evaluation Metrics :
SVM Classification sklearn
Evaluation Metric: Accuracy=0.5100, Precision=0.4642, Recall=0.5100, F1 Score=0.4585
Random Forest sklearn
Evaluation Metric: Accuracy=0.4900, Precision=0.4321, Recall=0.4900, F1 Score=0.3802

sl_liwc_tokenized  Sklearn Evaluation Metrics :
SVM Classification sklearn
Evaluation Metric: Accuracy=0.5233, Precision=0.4747, Recall=0.5233, F1 Score=0.4785
Random Forest sklearn
Evaluation Metric: Accuracy=0.5067, Precision=0.4164, Recall=0.5067, F1 Score=0.3853
```

3) Cross Validation Using Random Forest classification:

Filtered Set:

1) Preprocessed_fs:

```
______________________CROSS-VAL Filtered random_forest_classification SkLearn Classification______________________

preprocessed_fs Sklearn Evaluation Metrics :
Random Forests Classifier
Starting Fold 0
fold-0, Accuracy-0.515
Starting Fold 1
fold-1, Accuracy-0.465
Starting Fold 2
fold-2, Accuracy-0.55
Starting Fold 3
fold-3, Accuracy-0.465
Starting Fold 4
fold-4, Accuracy-0.475
mean accuracy- 0.49400000000000005
average precision 0.0
average recall    0.0
F-score           0.0
```

2) Sl_preprocessed:

```
sl_preprocessed Sklearn Evaluation Metrics :
Random Forests Classifier
Starting Fold 0
fold-0, Accuracy-0.51
Starting Fold 1
fold-1, Accuracy-0.46
Starting Fold 2
fold-2, Accuracy-0.465
Starting Fold 3
fold-3, Accuracy-0.495
Starting Fold 4
fold-4, Accuracy-0.505
mean accuracy- 0.487
average precision 0.4870000000000045
average recall    0.4109836122941413
F-score           0.4921927331651637
```

3) Pos_preprocessed:

```
pos_preprocessed Sklearn Evaluation Metrics :
Random Forests Classifier
Starting Fold 0
fold-0, Accuracy-0.48
Starting Fold 1
fold-1, Accuracy-0.44
Starting Fold 2
fold-2, Accuracy-0.495
Starting Fold 3
fold-3, Accuracy-0.415
Starting Fold 4
fold-4, Accuracy-0.43
mean accuracy- 0.45200000000000007
average precision 0.4520000000000018
average recall    0.3684203938115332
F-score           0.420724902366148
```

4) Liwc_preprocessed:

```
liwc_preprocessed Sklearn Evaluation Metrics :
Random Forests Classifier
Starting Fold 0
fold-0, Accuracy-0.525
Starting Fold 1
fold-1, Accuracy-0.545
Starting Fold 2
fold-2, Accuracy-0.5
Starting Fold 3
fold-3, Accuracy-0.52
Starting Fold 4
fold-4, Accuracy-0.55
mean accuracy- 0.5279999999999999
average precision 0.5280000000000074
average recall    0.46042655807280647
F-score           0.5461781383483115
```

5) Sl_liwc_preprocessed:

```
sl_liwc_preprocessed Sklearn Evaluation Metrics :
Random Forests Classifier
Starting Fold 0
fold-0, Accuracy-0.54
Starting Fold 1
fold-1, Accuracy-0.5
Starting Fold 2
fold-2, Accuracy-0.445
Starting Fold 3
fold-3, Accuracy-0.51
Starting Fold 4
fold-4, Accuracy-0.49
mean accuracy- 0.49700000000000005
average precision 0.49700000000000055
average recall    0.4403920909239355
F-score           0.48749056940617613
```

Unfiltered Set:

1) tokenized_fs:

```
______________________CROSS-VAL Unfiltered random_forest_classification SkLearn Classification______________________

tokenized_fs Sklearn Evaluation Metrics :
Random Forests Classifier
Starting Fold 0
fold-0, Accuracy-0.51
Starting Fold 1
fold-1, Accuracy-0.48
Starting Fold 2
fold-2, Accuracy-0.535
Starting Fold 3
fold-3, Accuracy-0.49
Starting Fold 4
fold-4, Accuracy-0.49
mean accuracy- 0.501
average precision 0.501000000000007
average recall    0.3764449020357833
F-score           0.41494695572773815
```

2) sl_tokenized:

```
sl_tokenized Sklearn Evaluation Metrics :
Random Forests Classifier
Starting Fold 0
fold-0, Accuracy-0.545
Starting Fold 1
fold-1, Accuracy-0.485
Starting Fold 2
fold-2, Accuracy-0.56
Starting Fold 3
fold-3, Accuracy-0.51
Starting Fold 4
fold-4, Accuracy-0.49
mean accuracy- 0.518
average precision 0.5179999999999941
average recall    0.4183956123195254
F-score           0.47295175844693477
```

3) pos_tokenized:

```
 pos_tokenized Sklearn Evaluation Metrics :
Random Forests Classifier
Starting Fold 0
fold-0, Accuracy-0.51
Starting Fold 1
fold-1, Accuracy-0.455
Starting Fold 2
fold-2, Accuracy-0.57
Starting Fold 3
fold-3, Accuracy-0.51
Starting Fold 4
fold-4, Accuracy-0.48
mean accuracy- 0.505
average precision 0.5049999999999977
average recall    0.3970643401177628
F-score           0.4388372057790412
```

4) liwc_tokenized:

```
liwc_tokenized  Sklearn Evaluation Metrics :
Random Forests Classifier
Starting Fold 0
fold-0, Accuracy-0.515
Starting Fold 1
fold-1, Accuracy-0.49
Starting Fold 2
fold-2, Accuracy-0.53
Starting Fold 3
fold-3, Accuracy-0.51
Starting Fold 4
fold-4, Accuracy-0.52
mean accuracy- 0.513
average precision 0.5130000000000063
average recall    0.41922181511566553
F-score           0.463915994422106З
```

5) sl_liwc_tokenized:

```
sl_liwc_tokenized  Sklearn Evaluation Metrics :
Random Forests Classifier
Starting Fold 0
fold-0, Accuracy-0.55
Starting Fold 1
fold-1, Accuracy-0.49
Starting Fold 2
fold-2, Accuracy-0.535
Starting Fold 3
fold-3, Accuracy-0.54
Starting Fold 4
fold-4, Accuracy-0.51
mean accuracy- 0.525
average precision 0.5249999999999965
average recall    0.4322177541259674
F-score           0.49008573036773845
```

Summary of Accuracies:

Feature Sets	Filtered feature set				Unfiltered feature set			
	Naïve bayes	SVM Classification	Random Forest	C.V. Random Forest	Naïve bayes	SVM Classification	Random Forest	C.V. Random Forest
Unigram	0.4833	0.4833	0.4833	0.494	0.49	0.4867	0.4833	0.501
SL	0.52	0.52	0.5067	0.487	0.4967	0.5267	0.5133	0.518
POS	0.4967	0.4833	0.4667	0.452	0.5133	0.48	0.49	0.505
LIWC	0.55	0.5267	0.52	0.5279	0.4867	0.51	0.49	0.513
SL_LIWC	0.5433	0.5267	0.5	0.497	0.51	0.5233	0.5067	0.525

Challenges:

1)**Hyperparameter tuning**: The hyperparameters must frequently be adjusted for machine learning models to function at their best. The models' accuracy could be constrained by improper hyperparameter tweaking. This is why we did not use hyperparameter tuning.

2)**Reading Test Data** - As previously indicated, we were not able to read the test data entirely because of the restrictions because the data file is quite huge in terms of data phrases. Because of this, we followed the same path as we did in class labs. We split the training data set into two parts, using the first to train my classifiers and the second to test my model. This resolved the crashing issue.

3)**Accuracy** - The accuracy wasn't what we had anticipated, but by using other classifiers, we were able to work within the same range of accuracy.

Conclusion:

This project exemplifies the use of machine learning algorithms and natural language processing techniques for sentiment analysis. It attempts to predict the sentiment polarity (positive or negative) of words by extracting pertinent characteristics from text data and training classifiers.

The above table showed improved accuracy with LIWC feature set compared to the previous feature sets. Naïve Bayes with LIWC Feature set achieved highest accuracy of 55% in predicting sentiments of the movie reviews. Also, combining liwc and SL feature sets with Naïve Bayes was advantageous as we were able to get an accuracy of 54.33% which is second highest overall.

Observation:

We found that applying just cross validation on Random Forest classifier was not as effective in improving our model efficiency in classifying movie reviews. We think that using hyperparameter tuning along with cross validation can be useful in improving efficiency.

Lessons Learned:

1) We found that it is better to anticipate what format of the parameters must be before creating any function. Also it is important to understand the data type and format of input and output of function.
2) We also discovered that difference effiency of Naïve Bayes, Sk Learn algorithms and CV were not significantly different with each other. The difference between highest and lowest accuracy of all was only 9.8%

Task Distribution:

Sahil:

- Debugging, alpha filter, regex_clean function, generation of three feature sets, function for evaluation metrics, model creation.

Harshit:

- Data Preprocessing, Tokenization and stemming, Documentation, generation of two feature sets, model creation.