**IST664 Natural Language Processing**

**Final Report**

**Kaggle Movie Reviews**

**Author: Yunkai Yao**

**Date: 4/28/2024**

**Introduction**

This dataset was produced for the Kaggle competition at https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews, which employs the sentiment analysis data from Socher et al. http://nlp.stanford.edu/sentiment/. The data originate from Pang and Lee's original movie review corpus, which was derived from reviews on the Rotten Tomatoes website. Socher's team employed a crowdsourcing approach to manually label all subphrases of a sentence with sentiment labels ranging from 0 to "negative," 1 to "somewhat negative," 2 to "neutral," 3 to "somewhat positive," and 4 to "positive."

The two files primarily utilized for classification and modeling are "train.tsv" and "test.tsv." In the majority of the training process, we employed the cross-validation technique. This approach facilitates the enhancement of the model's accuracy. Moreover, given that our model is constructed upon data samples, cross-validation enables the development of more effective classifiers.

Given that the training data encompasses 156,060 phrases, it is essential to sample the data. The technique employed for data sampling is random sampling. This random sampling technique samples the data based on limit values. The primary rationale for selecting this technique is that each phrase in the training data has an equal probability of being selected.

**Steps in categorization and sentiment analysis**

1. **Getting data from train.tsv**

# function to read kaggle training file, train and test a classifier

def processkaggle(dirPath,limitStr):

# convert the limit argument from a string to an int

limit = int(limitStr)

os.chdir(dirPath)

f = open('/Users/yaoyunkai/Desktop/IST 664/Final Project/FinalProjectData/kagglemoviereviews/corpus/train.tsv', 'r')

# loop over lines in the file and use the first limit of them

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])

1. **Randomize the data and select a certain number of phrases from the phrase data**

# 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')

for phrase in phraselist[:10]:

print (phrase)

#commandline interface takes a directory name with kaggle subdirectory for train.tsv and a limit to the number of kaggle phrases to use

#It then processes the files and trains a kaggle movie review sentiment classifier.

if \_\_name\_\_ == '\_\_main\_\_':

if (len(sys.argv) != 3):

print ('usage: classifyKaggle.py <corpus-dir> <limit>')

sys.exit(0)

processkaggle(sys.argv[1], sys.argv[2])

1. Tokenization

The phrases in the training data were tokenized using python's nltk package.

1. Preprocessing

**Function Conversion**

This is a pivotal stage in the construction of the model, as the model will be trained on the selected features. The model's features have been defined based on the sample. Additionally, each phrase in the sample will be designed based on these features. The technique employed is called "Bag of Words."

Bag of Words: Identify the words that have been used in the selected samples, and the set of these words becomes the model's features. Each phrase in the sample is then analyzed based on these features. To train our model, we employed two distinct types of features. These include all the words utilized in the phrases present in the sample, as well as the 500 most frequently utilized phrases in the sample.

**Feature Set Functions**

A variety of feature set functions were employed for the phrases in the sample, which were then provided to the model for training. The model used for training the samples is the Naive Bayes classifier in NLTK. The following is a list of the different types of function feature sets used in the 2000 samples.

1. Word Vector/Single Character Feature Set: This feature set converts the phrases in the sample into a vector defined by the presence or absence of a word. Each phrase has a true/false vector, and the length of the vector is equal to the length of the bag of words. These features are provided to NLTK's Naive Bayes classifier. Through cross-validation (n=5), the results (accuracy and confusion matrix) for the two different features are as follows:
2. Use all the words in the phrase as word features: ('mean accuracy', 0.531)
3. Characterize words by the 500 most common words in a phrase: ('mean accuracy', 0.5115000000000001)
4. NOT feature set: In addition to the single character features, NOT features are considered for each phrase in the sample. NOT features are negations of words that are followed by negations, such as not, never, etc. The NOT features are employed for the negation of words that are followed by negations, such as not, never, etc. The NOT features are utilized for the negation of words that are followed by negatives. The aforementioned features were provided to NLTK's Naive Bayes classifier, and through cross-validation (n=5), the results (accuracy and confusion matrix) for the two different features were as follows:
5. Use all the words in the phrase as word features: ('mean accuracy', 0.547)
6. Characterize words by the 500 most common words in a phrase: ('mean accuracy', 0.5115000000000001)
7. The bigram feature set considers combinations of bigrams in phrases together with non-bigram features. The Bigram Association Measures from the NLTK package were employed, and the chi test was used to determine the most important bigrams. These features were provided to NLTK's Naive Bayes classifier, and through cross-validation (n=5), the results (accuracy and confusion matrix) for the two different features were as follows:
8. Use all the words in the phrase as word features: ('mean accuracy', 0.5255)
9. Characterize words by the 500 most common words in a phrase: ('mean accuracy', 0.5065)
10. The POS Tag Feature Set includes adverbs, adjectives, verbs, and nouns. Additionally, the tags of each word in a phrase are considered to enhance the feature set. These features were provided to NLTK's Naive Bayes classifier using cross-validation (n=5), and the results (accuracy and confusion matrix) for the two different features are as follows:
11. Use all the words in the phrase as word features: ('mean accuracy', 0.5375)
12. Characterize words by the 500 most common words in a phrase: ('mean accuracy', 0.517)
13. Sentiment Dictionary: Subjectivity Count Feature. In this particular feature set function, two additional features are added to the unity feature: positive and negative counts. This feature set employs the definition of the readSubjectivity function in the sentiment\_read\_subjectivity.py module provided by the professor. It generates a subjectivity lexicon, represented here as a dictionary, where each word is mapped to a list containing intensity and polarity, which primarily define the way SL features are defined. The aforementioned features were provided to NLTK's Naive Bayes classifier, and through cross-validation (n=5), the results (accuracy and confusion matrix) for the two different features were as follows:
14. Use all the words in the phrase as word features: ('mean accuracy', 0.5405)
15. Characterize words by the 500 most common words in a phrase: ('mean accuracy', 0.532)

**comparisons**

Across the different feature set functions, Sentiment Dictionary and Non-Feature appear to perform better than the other functions in categorizing samples. According to the Kaggle competition leaderboard, the actual accuracy is approximately 62%, while our model performs similarly in terms of the subjectivity score of the Sentiment Dictionary. Additionally, the accuracy is higher when all words are considered compared to the most common 500 words. The model demonstrated satisfactory performance following training on a subset of the available data. However, it is evident that the accuracy of the model could be enhanced by training on the entire dataset. This is evidenced by the observation that the accuracy of the model appears to improve as the sample size increases.

**Training with Weka**

The CSV file "trainWeka.csv," created using unigram features, has been loaded into Weka and different classifier techniques have been applied to it. Previously, NLTK Naive Bayes was used to train the model. The techniques applied in Weka are J48, Ada Boost, Logistic Regression, and ZeroR. The sample size of these classifiers is 5000, and the word features used are 200. The outcomes of the classifier with the superior performance are stored in the "resultBuffers" folder, and a summary of the observations is as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Technique | Accuracy | Precision | Recall | F-measure |
| AdaBoost | 52.14% | 0.272 | 0.521 | 0.358 |
| J48 | 52% | 0.374 | 0.520 | 0.366 |
| Bagging | 48.24% | 0.347 | 0.482 | 0.375 |
| Logistic | 51.96% | 0.409 | 0.520 | 0.376 |
| Naive Bayes | 51.98% | 0.405 | 0.520 | 0.374 |
| ZeroR | 52.16% | 0.272 | 0.522 | 0.358 |

Among the various techniques employed for single string and word vector features, the ZeroR and AdaBoost techniques appear to exhibit slight superiority over the NLTK naive Bayes classifier. Consequently, when presented with the NOT feature set and SL features, the outcomes yielded by the NLTK Naive Bayes classifier may be enhanced.

**Summary**

Following the preprocessing of the data, the accuracy of our model increased by 3%, indicating that the preprocessing of phrases represents a crucial step in modeling. Additionally, among the various feature sets, the NOT features and SL features appear to yield superior results. It can be observed that, in addition to Naive Bayes, other techniques in Weka (e.g., AdaBoost and ZeroR) demonstrate superior performance in processing sample phrases. The optimal model achieved an overall accuracy of 56%, which is a satisfactory outcome.