Group Name: sPredict

Group Members: Anoop S Somashekar and Vikas Naidu

Problem Description

Given a data containing movie reviews, our model will predict the sentiment of the reviews. We will classify the sentiment as neutral, positive or negative. Our model will be as generic as possible i.e. reviews can be from various sources like product reviews, movie reviews, restaurant review. Our model will not consider the source of the review like movie, product etc. It only analyzes the review text to predict the sentiment.

Proposed Solution

We have implemented two solutions for the sentiment prediction problem to contrast between naïve strategy and improvised strategy. In the naïve strategy, we will count the number of positive words and number of negative words in a review. The one with highest count will be predicted as the sentiment of the review and equal counts or zero counts will be considered as neutral sentiment. In the improvised strategy, we will do feature engineering on the review data set which will be used by the machine learning model to predict the sentiment of the review.

Full Implementation Details

1) Preprocessing

We got the movie review dataset from the kaggle repository. Apart from review text and its sentiment, the dataset had some additional parameters which were not important for us. Hence we implemented a standalone java program which parses this dataset and generates a new file with only movie review and its sentiment. Also, the data had the sentiment labelled with 5 classes as below:

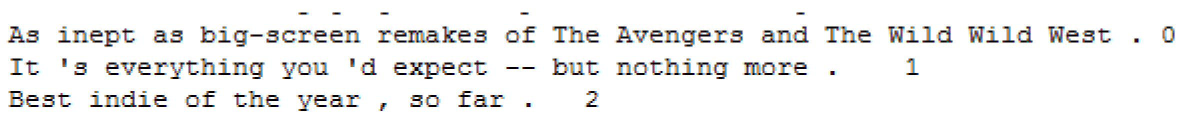
0 – negative, 1 - somewhat negative, 2 – neutral, 3 - somewhat positive and 4 – positive

So we scaled it down to 3 classes by merging 0, 1 as negative and 3, 4 as positive. The three classes for the sentiment are as below:

0 – negative, 1 – neutral and 2 – positive

(Final predicted labels will be like 'neg', 'neu' & 'pos' for negative, neutral & positive reviews respectively)

Sample input after preprocessing:



```
As inept as big-screen remakes of The Avengers and The Wild Wild West . 0
It 's everything you 'd expect -- but nothing more .    1
Best indie of the year , so far .   2
```

2) Baseline System

We obtained a list of positive and negative words from an online repository. We count the number of positive words in a review by checking if each token is present in the positive words list. Similarly, we count the number of negative words in a review. If a review has more positive words than negative words then we predict the sentiment of the review as positive. Similarly, if a review has more negative words than positive words then we predict the sentiment as negative. If a review has equal number of positive and negative words then we predict the sentiment as neutral.

We have used precision and recall as the evaluation metrics. Precision and recall is defined as below:
Precision = True positive count of a class / (True positive of the corresponding class + False positive count of the corresponding class)

Recall = True positive count of a class / (True positive of the corresponding class + False negative count of the corresponding class)

We computed the above parameters for precision and recall by comparing our predicted sentiment with the actual sentiment of the review.

Baseline Results:
Our dataset consisted of 1500 reviews. The accuracy obtained from the baseline system is as below:

```
****************************************************************
*          Baseline bag of words model Accuracy Results (1500 Reviews)          *
****************************************************************

Precision of label neg is 0.00 %
Precision of label neu is 30.44 %
Precision of label pos is 43.68 %

Recall of label neg is 0.00 %
Recall of label neu is 29.62 %
Recall of label pos is 84.64 %
```

3) Improvement Strategy

We have used the Bag-of-words model as a tool for feature generation. After transforming the review text from the corpus into a "bag of words", we then obtained various measures to characterize the text. The characteristics that were incorporated into our feature vectors were tokens, lemmas, POS tags, dependency graphs, synonyms and hypernyms. These characteristics which were nothing but two lexical, two syntactic and two semantic feature respectively formed six components of a review feature. We then normalized our review features so that each of their lengths could be made equal to the total number of unique features in our collection.

Each vector was represented as an array of numeric values as follows

```
(1) [1, 2, 1, 1, 2, 0, 0, 0, 1, 1,…........,3]
(2) [1, 1, 1, 1, 0, 1, 1, 1, 0, 0,…........,2]
```

The length of our feature vector after analyzing around 1500 hundred instances of review for six features was ~8000. It took less than 2 mins of processing for generating final vectors file. The length of our feature vector after analyzing full dataset which contained 8500 reviews was ~21000 and it took less than 10 mins of processing to generate final vector file.

It is important to note that although there were so many features, most of the feature values were zero and thus the final vectors file was a sparse matrix.

Lexical Features:

a) Tokenization and stop words removal:

We have used Stanford NLP TokenAnnotation and TextAnnotation classes to tokenize the reviews. Our dataset format is such that each line contains a single review. So we read one line at a time from the review file and Stanford Tokenizer will tokenize each review. We skip the token if it is present in the stop words list. All non-stop word tokens are added to the unique feature list. These tokens are further processed to generate other features like lemma, part of speech tag, synonymy, hypernymy etc.

b) Lemmatization

We used Stanford NLP LemmaAnnotations class to lemmatize each token. We then added the lemmatized word to the unique features list.

Syntactic Features:

c) Part of Speech Tagging

We used Stanford NLP PartOfSpeechAnnotation class to do part of speech tagging for each token. We then added the POS tag to the unique feature list.

d) Typed Dependencies

We used BasicDependenciesAnnotation class to generate dependencies for each review. It takes the entire review text as an input unlike other features. It generated various relations like nmod, amod, neg, advmod for the review tokens. We created a bigram for the relation neg with the governor word and added it to the unique feature list. For example for the relation neg (Entertain, not), we added bigram neg_Entertain to the unique feature list. We also counted number of neg relation in a review and added as a feature. We also considered amod and advmod relation and used the count of positive words and negative words as the feature. For Example, amod (performance, perfect) the dependent word perfect is positive and amod (ending, happy) the dependent word happy is positive so the count for the review would be 2. We also added the sentiment of the root word of typed dependencies as the feature. If the root word of the review is positive we added 1 as the feature, if the root word is negative then -1 else 0.

Semantic Features:

e) Synonymy

We used python nltk synset property to generate synonyms for each token and then select the first synonym in the returned list to add it to the unique feature list.

f) Hypernymy

We used python nltk hypernymy property to generate hypernyms for each token and then select the first hypernym in the returned list to add it to the unique feature list.

Improvised Strategy Results:

After adding the above mentioned features, our machine learning model with gradient boosting with 5 fold cross validation achieved the following accuracy:

```
 [1] "Accuracy of training data:"
[1] "Precision of class neg is 71.00 %"
[1] "Precision of class neu is 82.00 %"
[1] "Precision of class pos is 78.00 %"
[1] "Recall of class neg is 84.00 %"
[1] "Recall of class neu is 43.00 %"
[1] "Recall of class pos is 82.00 %"

[1] "Accuracy of test data:"
[1] "Precision of class neg is 53.00 %"
[1] "Precision of class neu is 29.00 %"
[1] "Precision of class pos is 65.00 %"
[1] "Recall of class neg is 69.00 %"
[1] "Recall of class neu is 10.00 %"
[1] "Recall of class pos is 72.00 %"
```

4) Examples

We generated the above mentioned features for each review text as shown below:

Review Text: A series of escapades demonstrating the adage that what is good for the goose is also good for the gander, some of which occasionally amuses but none of which amounts to much of a story.

Token: series, POSTag: NN, Lemma: series, Synonym: series, Hypernym: ordering

Token: escapades, POSTag: NNS, Lemma: escapade, Synonym: adventure, Hypernym: undertaking

Token: demonstrating, POSTag: VBG, Lemma: demonstrate, Synonym: show, Hypernym: show

Token: adage, POSTag: NN, Lemma: adage, Synonym: proverb, Hypernym: saying

Token: good, POSTag: JJ, Lemma: good, Synonym: good, Hypernym: advantage

Token: goose, POSTag: NN, Lemma: goose, Synonym: goose, Hypernym: anseriform_bird

Token: also, POSTag: RB, Lemma: also, Synonym: besides, Hypernym: null

Token: good, POSTag: JJ, Lemma: good, Synonym: good, Hypernym: advantage

Token: gander, POSTag: NN, Lemma: gander, Synonym: gander, Hypernym: goose

Token: occasionally, POSTag: RB, Lemma: occasionally, Synonym: occasionally, Hypernym: null

Token: amuses, POSTag: VBZ, Lemma: amuse, Synonym: amuse, Hypernym: entertain

Token: but, POSTag: CC, Lemma: but, Synonym: merely, Hypernym: null

Token: amounts, POSTag: VBZ, Lemma: amount, Synonym: sum, Hypernym: assets

Token: much, POSTag: RB, Lemma: much, Synonym: much, Hypernym: large_indefinite_quantity

Token: story POSTag: NN Lemma: story Synonym: narrative Hypernym: message

5) Programming Tools

The following programming tools and libraries were used during the implementation of our project

Stanford Core NLP - It is a natural language analysis tool implemented in Java programming language. We used 3.6.0 (latest) version of this tool for obtaining tokens, lemma and POS Tags of the tokens in our reviews. Also we used it to generate dependency graphs of review sentences. These four components comprised of two lexical, two syntactic features which were used to form feature vector for each review.

We choose Stanford core NLP because it provided us following benefits

- An integrated toolkit with a good range of grammatical analysis tools
- Fast, reliable analysis of arbitrary texts
- The overall highest quality text analytics

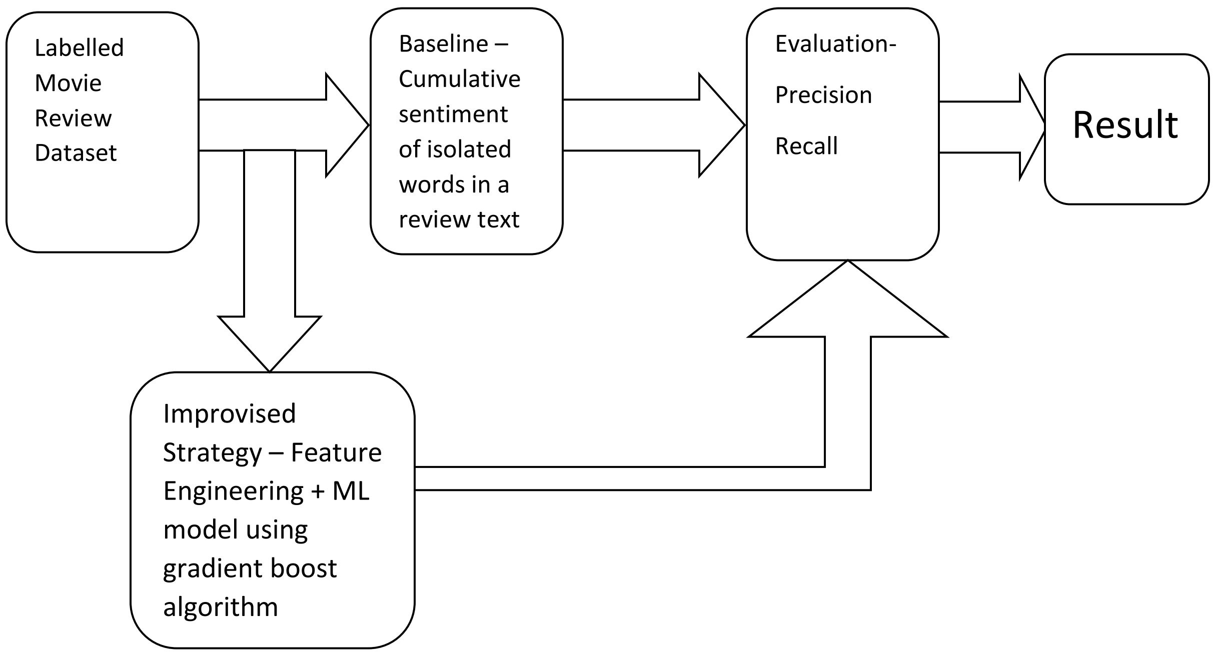
Usage: Stanford core NLP allowed us to define a single pipeline in Java for all above mentioned tasks-

```
Properties props = new Properties ();

props.setProperty ("annotators”, “tokenize, ssplit, pos, lemma, parse");

StanfordCoreNLP pipeline = new StanfordCoreNLP (props);
```

NLTK 3.0 - It is a natural language toolkit written in python programming language and provides easy access to WordNet resources. We used NLTK in order to obtain synonyms and hypernyms for tokens in our reviews. These two feature provide semantic information about reviews and hence two very important components of feature vector.

R – We used R programming environment to implement machine learning pipeline for the feature vectors.

6) <u>Architectural Diagram</u>


Labelled Movie Review Dataset
Baseline – Cumulative sentiment of isolated words in a review text
Evaluation- Precision Recall
Result
Improvised Strategy – Feature Engineering + ML model using gradient boost algorithm

7) Analysis

It can be observed from the accuracy results of baseline strategy that it performed very badly for the negative sentiment reviews i.e. it achieved 0% precision. The negativity of the review is not only based on negative words but it can also be of the form where positive word is negated. As our baseline strategy only considers majority of negative and positive words, it is obvious that it cannot predict the negative sentiment of the review. But it was a good way to start with as it helped us to understand the various scenarios in predicting the sentiment of the reviews.

We started added new features to improve the accuracy of our model. First we added 3 new features namely tokenize, lemma and part of speech using count bag of words model and achieved the following accuracy. Note we have only analyzed the accuracy of unseen test data as the machine learning algorithm can over-fit with the training data. Hence it is not considered for the analysis.

```
*************************************************************
*          3 Feature Accuracy Results (1500 reviews)          *
*************************************************************

 [1] "Accuracy of test data:"
[1] "Precision of class neg is 43 %"
[1] "Precision of class neu is 29 %"
[1] "Precision of class pos is 49 %"
[1] "Recall of class neg is 39 %"
[1] "Recall of class neu is 6 %"
[1] "Recall of class pos is 69 %"
```

These 3 new features did improve our accuracy as compared to the baseline strategy. Even though we have not considered contextual meaning of the review and we have not performed specific handling for the negation, it performed well. The reason for this could be that the machine learning algorithm was able to find some pattern by looking at these newly added features and hence it improvised our prediction.

Next we added 2 more features namely synonymy and hypernymy to the already existing 3 features. The accuracy results with 5 features is as below:

```
****************************************************************

*         5 Feature Accuracy Results (1500 reviews)            *

****************************************************************

 [1] "Accuracy of test data:"

[1] "Precision of class neg is 46.00 %"

[1] "Precision of class neu is 29.00 %"

[1] "Precision of class pos is 52.00 %"

[1] "Recall of class neg is 57.00 %"

[1] "Recall of class neu is 9.00 %"

[1] "Recall of class pos is 62.00 %"
```

We can observe slight improvement in the accuracy of the 5 feature model as compared to the 3 feature model. The new features might have helped the ML algorithm to find the commonalities of the reviews of similar type with different words usage in the reviews. For example, 2 different reviews with words good and magnanimous respectively was considered as different in the earlier model will predict as similar with synonymy and hypernymy features.

We then generated typed dependencies features for each review and added various features based on the dependency relation. This model performed slightly better than 5 feature model. The accuracy results for 6 feature model is as below:

```
****************************************************************

*         6 Feature Accuracy Results (1500 reviews)            *

****************************************************************

[1] "Accuracy of test data:"

[1] "Precision of class neg is 53.00 %"

[1] "Precision of class neu is 29.00 %"

[1] "Precision of class pos is 65.00 %"

[1] "Recall of class neg is 69.00 %"

[1] "Recall of class neu is 10.00 %"

[1] "Recall of class pos is 72.00 %"
```

It can be observed from all the above accuracy results that the accuracy of positive and negative sentiment is improving on adding new features whereas none of the model could improvise the prediction of neutral sentiment. The reason for this could be we have not extracted the contextual information from the review. We are generating features at the token level and only typed dependencies feature considered some relation among the tokens in the review. It can be concluded that predicting the neutral sentiment review is more complex than positive and negative sentiment. So we need to add more advanced features with contextual information to further improvise our model.

8) Potential Improvements

We could add more features like n grams, shingling and consider other typed dependencies relations like nmod, dobj, nsubj to generate new features. Since our feature size was very big, we could only test with 1500 reviews. The machine learning model might need more data to learn better. Hence providing more training data with sufficient distribution of data for each class might improve the accuracy.