Sentiment Detection of Reviews

Author: Yeziwei Wang

1 Introduction

This report explores different methods for sentiment detection of movie reviews. Initially, a non-machine learning approach was explored using sentiment lexicon. Sentiment detection was then conducted using machine learning methods: Naive Bayes classifier and Support Vector Machine using bag-of-words representation. The result from NB classifier is used as a baseline for other machine learning methods. Cross-validation is applied to avoid over-fitting. Finally, a shallow neural network is used to produce a set of word vectors and paragraph vectors. These are then used as features to detect sentiment in reviews.

There are two sets of data used. The first dataset contains 1000 positive and 1000 negative reviews. This is used in sentiment lexicon method, NB classifier and SVM. The second dataset is the IMDB sentiment dataset which contains 25,000 labelled positive and negative reviews, as well as 50,000 unlabelled reviews. This data is used to train Doc2Vec models. The first data set is then used to test these Doc2Vec models with SVM.

2 Sentiment Lexicon

Two methods are used for sentiment detection based on a sentiment lexicon: Binary and Weighted solutions. The sentiment lexicon is a dictionary that contains information of words, like polarity, type etc. Polarity determines if the word is positive or negative. Type shows the degree of polarity. If type is 'strongsubj', the word is strongly subjective (higher weights) and if the type is 'weaksubj', the word is weakly subjective (lower weights)(Riloff and Wiebe, 2003). In the Binary approach, only the counts of positive and negative words are considered. A threshold is used to classify the sentiment of a review.

```
if positive - negative > threshold:
    result = "POS"
else:
    result = "NEG"
```

For the Weighted approach, both polarity and type of words are considered. More weights are added to a word with strong type and less to a word with weak type. In the code implementation, a weak word scores one whereas a strong word scores three.

Scores	Binary	Weighted	
		strong	weak
Positive	1	3	1
Negative	1	3	1

Table 1: Word Scoring

	Token-only	Maginitude
Accuracy	0.68	0.69

Table 2: Prediction using Sentiment Lexicon

Q0.1

From Table 2, using magnitude improves accuracy by 1%, which is not significant. Since weighted solution generates higher scores, the threshold should be at least more than that of binary method. After testing accuracy by increasing threshold from 8 to 30, a threshold of 8 gives the highest accuracy. Therefore, the threshold is chosen to be 8 for both methods.

Q0.2

The performance difference of the two methods is not statistically significant. This shows that in sentiment analysis, the magnitude of words doesn't provide much extra information.

In general, the accuracy achieved by sentiment lexicon analysis is poor. This is because we only consider the polarity from the lexicon, which is out of context. A negative word may indicate positive meaning in a phrase or a sentence(Wilson, Wiebe and Hoffman, 2005). Also, simply consider the counts of positive or negative words can neglect the actual meaning of the review. For example, a review can contain many positive words and end with a strongly negative sentence with 'however'.

3 Machine Learning using Bags of Words representations

3.1 Naive Bayes Classifier

In Naive Bayes Classifier, the best class is determined by the maximum likelihood.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|\bar{f}) = \underset{c \in C}{\operatorname{argmax}} P(c) \Pi_{i=1}^{n} P(f_i|c)$$

$$P(c) = \frac{N_c}{N} \qquad\qquad P(f_i|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

In the simple NB classifier, if a word in the test data didn't occur in the training data then that word is ignored. Otherwise, it will lead to zero conditional probability and negative infinity in log-likelihood.

```
for term in self.vocabulary:
    if N_t_pos[term]:
        self.condProb["POS"][term]= float(N_t_pos[term])/len(text_pos)
    if N_t_neg[term]:
        self.condProb["NEG"][term] = float(N_t_neg[term])/len(text_neg)
```

Q1.0

The accuracy of Naive Bayes on the held-out test set is 0.46. One reason for such low accuracy may be that there are many new words in test reviews and all them are discarded. On the other hand, words like 'the' and 'a' are most likely to appear in all reviews, but they don't provide sentiment information.

Smoothing

Smoothing is used to deal with new words in test set. In this example, add-one smoothing is used. Therefore, the conditional probability becomes:

$$P(t|c) = \frac{T_{ct}+1}{\sum_{t' \in V}(T_{ct'}+1)} = \frac{T_{ct}+1}{(\sum_{t' \in V} T_{ct'}) + B'} \quad where\, B'\, is\, the\, vocabulary\, size. \tag{1}$$

Q2.0

```
if self.smoothing == True:
    for term in self.vocabulary:
        self.condProb["POS"][term] = float(N_t_pos[term]+1)/(len(text_pos)+len(self.vocabulary))
        self.condProb["NEG"][term] = float(N_t_neg[term]+1)/(len(text_neg)+len(self.vocabulary))
```

The accuracy after smoothing is 0.82. Smoothing significantly improved the accuracy of the classifier.

Q2.1

For Naive Bayes classifier, the difference between predictions with and without smoothing is statistically significant.

This shows that the words discarded in NB without smoothing actually contain a lot of information. In NLP, the problem is obtaining enough suitable training data. We cannot always have all the words in the training data, thus it is important for a model to deal with new data properly. There are two possible ways to improve the performance of Naive Bayes classifier. 1) From other research, it seems that binary multinomial Naive Bayes works better for sentiment analysis. Binary NB only considers the presence of a word rather than its counts. 2) Improvements on dealing with negation: prepending prefix $NOT\_$ to every word after a logical negative word, such as 'not', 'no', 'never', until the next punctuation (Stanford NLP Group).

Due to above performance, smoothing is used in the following Naive Bayes classifier modifications.

3.2 Cross-validation

Q3.0

```
for i in range (0,10):
    test_files, train_files = [], []     # reset files
    for j in range (0,10):
        if j==i:
            test_files = corpus.folds[j] # change test set for each validation
        else:
            train_files.extend(corpus.folds[j])
```

The 10 accuracies from 10-folds cross-validation is shown in Table 3. Cross-validation can test if the model is over-fitting the training data. From the results, the accuracies of the 10-folds are similar with average accuracy of 0.81. The standard deviation is 0.03 showing that the testing result is confident and not over-fitting.

	Uni-gram & Non-stemmed		Uni-gram & Stemmed		Bi-gram & Non-stemmed		Tri-gram & Non-stemmed	
	Accuracy	Feature size	Accuracy	Feature size	Accuracy	Feature Size	Accurace	Feature size
1	0.77	51854	0.78	31897	0.77	531431	0.77	970622
2	0.83	51986	0.81	31992	0.83	532895	0.83	972066
3	0.82	51790	0.83	31847	0.84	531630	0.83	971174
4	0.83	52119	0.83	32049	0.85	533145	0.82	972224
5	0.79	51903	0.80	31951	0.81	530069	0.80	966260
6	0.84	51725	0.80	31879	0.82	527972	0.80	961662
7	0.81	51548	0.83	31727	0.82	527125	0.81	961068
8	0.81	51592	0.81	31736	0.82	528794	0.83	964938
9	0.77	51835	0.76	31837	0.81	530312	0.79	967546
10	0.82	51699	0.82	31737	0.84	529419	0.82	964943
Average	0.81	51805	0.81	31865	0.82	530279	0.81	967250
Std Dev	0.03		0.02		0.02		0.02	

Table 3: Naive Bayes Classifier 10-folds cross validation result

3.3 Features, overfitting, and the curse of dimensionality

3.3.1 A Touch of Linguistics

	Non-stemming	Stemming
Accuracy	0.82	0.82
Feature Size	51699	31737

Table 4: Results from Held-out test set

Q4.0

After stemming, the 10-folds cross validation average accuracy is 0.81 from Table 3. The performance stays the same as before stemming. This is because a positive word stays positive after stemming, which doesn't provide extra information for sentiment analysis. In NB classifier, the

total counts of words having the same stem is the same as the counts of the stem in a stemmed document. Thus, both have the same conditional probability.

Q4.1

The performance difference of stemming and non-stemming is not statistically significant.

Q4.2

Number of features reduced significantly after stemming. From table 3, the average vocabulary from non-stemmed corpus is 51,805 and the average vocabulary from stemmed corpus is 31,865. In Table 4, the results from the held-out test set is similar: feature size decreased from 51,699 to 31,717. The number of features is vocabulary size in NB model, stemming makes multiple words arrive at the same stem, which reduces the number of unique words in the corpus. Smaller feature size can reduce computation complexity.

Since stemming didn't improve the accuracy, the following training continue to use non-stemmed corpus.

3.3.2 Word Order

Word order information can be retained by using bi-gram and tri-gram models. In bi-gram, a feature is a word sequence of two coherent words. In tri-gram, a feature is a sequence of three words. Therefore, bi-gram and tri-gram model should have much larger feature size. In Table 3, the results are obtained from uni-gram + bi-gram and uni-gram + tri-gram.

	Uni-gram	Uni-gram+Bi-gram	Uni-gram+Tri-gram
Accuracy	0.82	0.84	0.82
Feature Size	51,699	529,419	964,943

Table 5: Results from Held-out test set

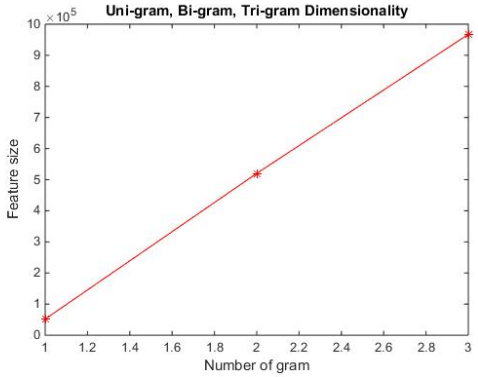
Q5.0

The 10-folds cross validation results are in Table 3. The average accuracy for using bigram as features is 0.82 with standard deviation of 0.02. The average accuracy for using trigram as features is 0.81 with standard deviation 0.02. The performance difference is not statistically significant compared to NB classifier.

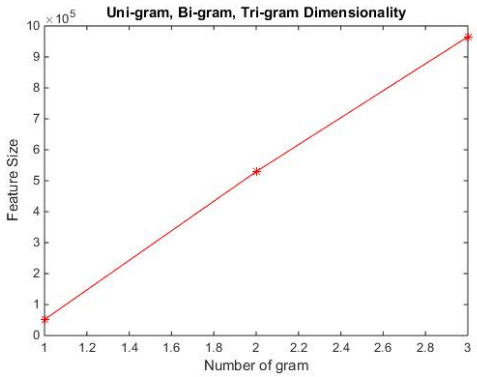
Q5.1

Bag-of-n-grams suffer from data sparsity and high dimensionality. A high-dimensional representation tends to generalise poorly. From the results in Table3, the feature size of a uni-gram model is 51,805, uni-gram + bi-gram model is 530,279 and uni-gram + tri-gram model is 967,250. The

feature size seems to increase linearly (Figure 1). The results from held-out test is similar, shown in Table 5.



(a) 10-folds Cross-validation Results



(b) Held-out Test Results

Figure 1: Feature Size Plot

3.4 Feature independence, and comparing Naive Bayes with SVM

NB classifier assumes features to be independent given document d. However, in reality features are not independent from each other. If the correlation is strong, NB performance can be skewed. On the other hand, Support Vector Machine classifier doesn't assume feature independence.

SVM	Token only	POS-tag	Discarding closed-class words
Folds	Accuracy	Accuracy	Accuracy
1	0.77	0.76	0.83
2	0.77	0.77	0.82
3	0.70	0.72	0.78
4	0.74	0.77	0.86
5	0.76	0.75	0.82
6	0.79	0.77	0.84
7	0.83	0.84	0.87
8	0.72	0.72	0.82
9	0.76	0.74	0.84
10	0.74	0.74	0.86
Average Accuracy	0.76	0.76	0.83
Std Dev	0.03	0.03	0.03

Table 6: Predictions with SVM

Q6.0

```
write_file = open(filename, "w")
for label, content in data:
    write_file.write(label+"\t")
    write_file.write("\t".join('%s:%s' %feature for feature in content)+"\n")
write_file.close()
```

The data file for SVM starts with labels of each review followed by tokens from each review.

Q6.1

The settings are kept as default when using SVM Light. The cross-validation results are shown in the first column in Table 6. The average accuracy is 0.76 and standard deviation is 0.03. The accuracy is much lower than NB classifier. This may be because linear separation is used in the SVM training, which may not the best kernel for the task.

Q7.0

When considering POS tags in the features, the average accuracy from cross-validation is 0.76, same with token-only accuracy, as well as standard deviation. From the result, POS tag didn't help with sentiment classification. This may be due to linguistic features. For semantic analysis, the difference in POS tags can completely change the meaning of a sentence. However, in sentiment analysis, when the same word have different POS tags, its sentiment content remains the same. For exampleTherefore, no extra information is provided.

Q7.1

After discarding all closed-class words, the average accuracy increased to 0.83 with the same standard deviation. The performance is statistically significant.

A closed-class barely acquire new members. In this case, discarding closed-classed words means only none, adjective, adverb, and verb are included in the dataset. The improvement in the accuracy shows that closed-class words, such as pronouns and conjunctions, don't provide any information for sentiment analysis. In a text, the sentiment of open-classed words, verbs like 'love', 'hate', adjectives like 'great', 'poor', determines the sentiment of the text. Discarding closed-class words removes noise and uncertainty in the document.

4 Extension

Q8.0

Five different models are trained using gensim Doc2Vec. 1) Distributive Memory using concatenation of context word vectors(in the report it is referred to as DMC), 2) Distributive Memory using mean of context word vectors (referred to as DMM), 3) Distributive Bag of Word(DBOW), 4) concatenation of DMC and DBOW, 5) concatenation of DMM and DBOW. The following settings are used for the model training:

Model settings	dm	size	window	min_count	hs	dm_concat	dm_mean	negative
DMC	1	100	10	3	1	1	0	8
DMM	0	100	10	3	1	0	0	8
DBOW	1	100	10	3	1	0	1	8

Table 7: Doc2Vec Model Setting

The feature size is chosen as 100, window size as 10 based on previous study (Mikolov 2014) and online gensim tutorial. Also, hierarchical softmax proved to be more efficient in practice. All 100,000 data in IMDB data set is used to train the models, including both labelled and un-labelled. One of the advantages of Doc2Vec model training is that the un-labelled data can also be used. The initial learning rate is set to be 0.025 and minimum learning rate is 0.001. Five epochs is used within each model with linearly decreasing learning rate. Five passes of training have the same initial alpha. Increasing number of passes didn't improve performance but drastically increased the training time. Therefore, the final number of passes and iterations are both set as 5.

```
for epoch in range(passes):
    shuffle(doc_list)
    for name, train_model in models_by_name.items():
        train_model.alpha, train_model.min_alpha = alpha, min_alpha
        train_model.train(doc_list, total_examples=len(doc_list), epochs=5)
```

Feature vectors are then inferred from the models trained and normalised using Euclidean distance:

$$\sqrt{\sum_{m=1}^{M} V_m^2(d)}$$

DMC takes the longest to train, because concatenation results in a much larger model including both word vectors and softmax weights (Lee and Mikolov, 2014). Whereas, DBOW trains really fast because it only stores softmax weights.

The accuracies of DBOW, DBOW+DMC and DBOW+DMM are statistically significant compared to the accuracy of 0.81 from NB classifier. Performance of DMC is lower than NB by 1%, and performance of DMM is greater by 3%. Both performance differences are not statistically significant. The best accuracy is produced by DBOW ignoring word orders. On the contrary, DMM and DMC, considering word orders, have lower accuracy. This result is consistent with the result from NB bi-gram and tri-gram model. In both cases, the accuracy didn't improve when word order is considered. However, the computing cost increased greatly.

Folds	NB	DMC	DMM	DBOW	DBOW+DMC	DBOW+DMM
1	0.77	0.78	0.85	0.89	0.86	0.84
2	0.83	0.78	0.81	0.9	0.9	0.91
3	0.82	0.8	0.82	0.86	0.88	0.86
4	0.83	0.79	0.84	0.91	0.92	0.89
5	0.79	0.82	0.87	0.9	0.89	0.89
6	0.84	0.81	0.87	0.9	0.9	0.88
7	0.81	0.79	0.82	0.9	0.87	0.85
8	0.81	0.86	0.81	0.89	0.91	0.88
9	0.77	0.78	0.8	0.86	0.85	0.87
10	0.82	0.81	0.86	0.91	0.91	0.89
Average	0.81	0.8	0.84	0.89	0.89	0.88
Std Dev	0.03	0.03	0.03	0.01	0.02	0.02

Table 8: 10-folds Cross-validation with Doc2Vec features

5 Conclusion

From all the methods above, only using sentiment lexicon gives the worst result. The methods using machine learning improves the accuracy in general. As a whole, the document vector representation produces better accuracy than bag-of-words representation. Between SVM and NB classifier, NB produces higher accuracy. However, after discarding closed-class words, SVM gives higher accuracy.

Bag-of-words representation doesn't consider the semantics of words, whereas Doc2Vec has a better representation of word semantics. For example, 'amazing' and 'wonderful' are close in the vector space compared to 'amazing' and 'awful'. These information produces better feature vectors for sentiment analysis. SVM using DBOW model gives a significantly higher accuracy compared to bag-of-word feature vector. Also, the Doc2vec models are supported by a much larger data set.

In terms of word ordering, for both NB and Doc2vec methods, word order didn't improve accuracy. Instead, increasing length of word window dramatically increased computing complexity for both cases.

In conclusion, word vector is a better representation of data in terms of semantics and sentiment. It does require sufficient training data, but gives better overall performance.

References

[1] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann (2005). *Recognizing Contex-tual Polarity in Phrase-Level Sentiment Analysis.*. Proc. of HLT-EMNLP-2005.

[2] E. Riloff and J. Wiebe. 2003. *Learning extraction patterns for subjective expressions.* In EMNLP-2003.

[3] Wang, Sida and Manning, Chris D. *Baselines and bigrams: Simple, good sentiment and text classification.* In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, 2012.

[4] Quoc V. Le, Tomas Mikolov (2014). *Distributed Representations of Sentences and Documents. arXiv.*.

[5] Jurafsky, D. and Martin, J. (2017). *Chaper 6, Speech and language processing.*. [India]: Dorling Kindersley Pvt, Ltd.

[6] Gensim, models.doc2vec tutorials, `https://radimrehurek.com/gensim/models/doc2vec.html#id2`

[7] Gensim Doc2Vec Tutorial on IMDB Sentiment Dataset, `https://github.com/RaRe-Technologies/gensim/blob/aaab5b6c54d947a93980b8ce34d43b894cfdbb89/docs/notebooks/doc2vec-IMDB.ipynb`