Yelp Reviews Sentiment Analysis

Julie Jiang

December 23, 2016

Abstract

The growing amount of information *encoded* in natural language flowing on the internet is placing more and more emphasis on meaningful information retrieval. For this project, we consider the sentiment classification at the document level. Using Yelp reviews as data, we trained a Maximum Entropy model to classify a text review as positive, objective or negative. With the aid of outside resources such as NLTK, Stanford NLP Part-of-Speech Taggers, and SentiWordNet, we achieved an overall accuracy of 80%, using primarily bag-of-words features.

1 Introduction

1.1 Background

Yelp is one of the most popular business rating websites that focuses mainly on restaurants. Customers can leave text reviews on a restaurants' Yelp page along with a star-rating. With the huge collection of text reviews collected by Yelp, it is interesting to see if we can train a model to classify a text review as positive, negative, or somewhere in between - objective. This is more commonly known as sentiment analysis in the field of natural language processing. While this sentiment classifier is only trained and tested on Yelp Reviews, it can be easily extended to apply to any other type of reviews that may embody sentiments.

"Excellent food. Superb customer service. I miss the mario machines they used to have, but it's still a great place steeped in tradition." – Positive

"PROS: Italian hoagie was delicious. Friendly counter employee. The restaurant was clean and neat. CONS: The pizza was not good. Pre-formed crust, NOT fresh dough. The price of the failure of a pizza WAS NOT CHEAP EITHER. I guess the name says it all. Get the hoagie, pass on the pizza." – Objective

"Wing sauce is like water. Pretty much a lot of butter and some hot sauce (franks red hot maybe). The whole wings are good size and crispy, but for $1 a wing the sauce could be better. The hot and extra hot are about the same flavor/heat. The fish sandwich is good and is a large portion, sides are decent." – Negative

1.2 Goal

Our goal is to employ supervised machine learning to train a Maximum Entropy model on a training set, and evaluate how well it performs on a test set as compared to the gold standards.

In our case, the gold standards are the actual classification labels of the text reviews. Given a text review, this model will be able to extract relevant features from it and choose a classification label that maximizes its posterior probability.

1.3 Data

The data set used throughout this project is obtained from Yelp, distributed as part of their Yelp Dataset Challenge[1]. The original dataset contains more than 2.5 million reviews. However, due to the time frame of this project, it was not feasible to utilize all of the existing reviews. Instead, among these, 40 000 reviews were selected to be the training set, 4 000 development set, and 4 000 test set, maintaining an 80% training and 20% validating ratio.

Each text review has an associated five-star rating, with 1 being the most negative and 5 the most positive. To reduce the dimensionality of the classification problem, we simplify the 5-class classification problem into three classes: positive, and negative and objective. Text Reviews with 4 or 5 stars are considered positive, 3 stars objective, and 1 or 2 stars negative. In each set of data, there is a roughly equally number of positive, objective, and negative reviews.

1.4 Literature Survey

Sentiment analysis has always been a hot topic in the natural language processing. Many of the existing papers on sentiment analysis cites back to [5], which performed an exemplary extensive sentiment analysis of movie reviews, and also was the source of inspiration for many feature extraction techniques used in this project. Negation detection is also a much discussed topic in natural language information retrieval, for which [2] and [10] did an extensive study on. Another important tool used in this project is SentiWordNet, a lexical resource for opinion mining, which was applied in [4] to the problem of sentiment analysis in film reviews.

1.5 Source Code

The are two python scripts written for this project. One contains modules that extract features (`FeatExtractor.py`), and one contains the classifier module (`SentimentClassifier.py`. Please refer to the source code or see Appendix A for usage documentation.

1.6 Technical Details

All files provided:

```
FeatExtractor.py
SentimentClassifier.py
models.tar.gz  # A compressed file of selected trained models
sample.json    # A sample of the data
Yelp Reviews Sentiment Analysis.pdf
```

Note that while all the text reviews in `sample.json` are taken from the training set, the classification labels in `sample.json` do not reflect their actual classification label. In fact, all of the sample text reviews are supposed to be positively labeled reviews.

[1] https://www.yelp.com/dataset_challenge

2 Data Preprocessing

In order to extract lexical and syntactic features from the datasets, we need to first preprocess all datasets. Preprocessing is done in two steps. First, all reviews are tokenized with NLTK's tokenizer (`word_tokenize`). Second, all reviews are tagged with a Part-Of-Speech (POS) from the Penn Treebank tagset using the Stanford Log-Linear Part-Of-Speech Tagger [8] with a bidirectional English model[2]. In the process, reviews that are not in English could not be successfully tagged with POS, and are therefore removed from the datasets. The following example sentence from a review

> *My wife's short rib ravioli was creative and very tasty.*

is preprocessed into the following:

```
[["My", "PRP$"], ["wife", "NN"], ["'s", "POS"], ["short", "JJ"],
 ["rib", "NN"], ["ravioli", "NN"], ["was", "VBD"], ["creative", "JJ"],
 ["and", "CC"], ["very", "RB"], ["tasty", "JJ"], [".", "."]]
```

3 Evaluation

To measure the performance of our model, we compute a few evaluation metrics relevant to information retrieval [7]. In addition to the two common metrics precision and recall, we also compute the F-measure [6] and the accuracy. In mathematical terms, for each classification label i,

$$\text{Precision}_i = \frac{tp_i}{tp_i + fp_i} \tag{1}$$

$$\text{Recall}_i = \frac{tp_i}{tp_i + fn_i} \tag{2}$$

$$\text{F-Measure}_i = \frac{2 \cdot \text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{3}$$

$$\text{Accuracy}_i = \frac{tp_i + tn_i}{tp_i + tn_i + fp_i + fn_i} \tag{4}$$

where tp are the true positives, fp - false positives, tp - true positives, and tn - true negatives. The F-measure effectively combines precision and recall metrics into a single metric, and the averaged accuracy is a good overall estimate of the performance of a model.

4 Maximum Entropy

Maximum Entropy models are frequently used in natural language processing for classification tasks [3]. It estimates the posterior probability that a review y is of classification label i given its feature sets x as:

$$p(i|x) = \frac{\exp(w_i \cdot f_i)}{\sum_{i' \in I} \exp(w'_i \cdot f_i)} \tag{5}$$

where w_i, f_i are weight and feature vectors, respectively. A feature $f_n(i, x)$ is

$$f_n(i, x) = \begin{cases} 1 \text{ if feature\_items} = \text{feature\_value and feature\_label } = i \\ 0 \text{ otherwise} \end{cases} \tag{6}$$

[2] english-bidirectional-distsim.tagger - trained using a bidirectional architecture and including word shape and distribution similarity features.

For example,

$$f_n(i,x) = \begin{cases} 1 \text{ if } \texttt{(u'unacceptable', u'JJ')} = 0.75 \text{ and label} = \texttt{u'positive'} \\ 0 \text{ otherwise} \end{cases}$$

Using the Natural Language Toolkit (NLTK) for Python, we trained a Maximum Entropy classifier that has its weights chosen to maximize entropy while remaining empirically consistent with the training set. The basis of the model is NLTK's `MaxentClassifier`, which is then wrapped inside another module `SentimentClassifier`[3] specifically for our purposes.

5 Feature Extraction

5.1 What Matters

One important criteria in selecting which features out of all extracted features to encode is their tf-idf scores, or term-frequency versus inverse-document-frequency score. The usual formula of tf-idf measures the importance of a word to a particular document in a collection of documents, rewarding words that appear frequently in a document but infrequently in the collection of documents. However, as this is a classification problem, we are interested in features that are important to a particular classification label. Therefore, we want to reward features that appear relatively frequently in reviews of a particular label but relatively infrequently across all reviews. To do so, we introduce a slight twist to the usual tf-idf formula. For each classification label i and feature f,

$$\text{tfidf}(f,i) = \text{tf}(f,i) \cdot \text{idf}(f,i) \tag{7}$$

where

$$\text{tf}(f,i) = \frac{\text{Number of reviews labeled } i \text{ that contains } f}{\text{Number of reviews labeled } i} \tag{8}$$

$$\text{idf}(f,i) = \log \frac{\text{Total number of reviews}}{\text{Number of reviews that contains} f} \tag{9}$$

Note that words that appear more than once in a single review are only counted once. This is because we are not concerned with the frequency of a feature's existence, just so long as it *does* exist.

As an example, consider a model that extracted only two features: the existence of the words *the* and/or *awesome*. The determiner *the* appears very frequently among all reviews, but it doesn't appear particularly frequent in any specific classification of reviews. Therefore, *the* is not a very informative feature. On the other hand, the adjective *awesome* may not appear as frequently as *the* does overall, but it appears to be more important to the classification *positive*.

	Raw Counts			**tf-idf Scores**		
	Positive	**Objective**	**Negative**	**Positive**	**Objective**	**Negative**
the	11509	12055	11953	0.103	0.107	0.107
awesome	965	510	224	0.229	0.121	0.053

Table 1: The tf-idf scores of *the* and *awesome* based on their raw counts of occurence in the training set.

This tf-idf formula precisely captures this information. It assigns a higher to score to (*awesome*, *positive*) than it does to any other feature and label combination. Hence, the existence of the word *awesome* is a good predictor that a review is *positive*.

[3]See `SentimentClassifier.py`

5.2 A Spectrum of Sentiment

Since our aim is to classify a review into one of three possible classification labels, it seems intuitive that we should train a multiclass MaxEnt classifier. However, there is an important underlying property of objective sentiments, and that is it lies *between* the two opposite polars of positive and negative sentiments. Therefore, a second way we can go about this is to train a binary MaxEnt classifier and classify those reviews that have a nearly equal probability of being positive and negative as objective. This reduces the complexity of the classification problem while still preserving the identities of the three classifications.

For some of the features that we will experiment with, we will train both a binary model and a trinary model to see which performs better.

5.3 Bag-of-Words

A common feature used in sentiment analysis is bag-of-words. That is, to simplify the representation of each review into a bag-of-words that appeared in the review. Each word is a feature, which has a value of 1 if that word occurred in the review, and 0 otherwise. The same idea can be extended to not only unigrams but any number of grams. However, while a larger n-gram can capture more contextual information, it risks losing generality and is too prone to overfitting. Therefore, we will consider only unigrams, bigrams, and trigrams. As outlined in 5.1, we will extract only features that have a significant tf-idf score – a maximum tf-idf score that exceeds a certain threshold. For example, considering the following sentence from a negatively labeled review:

> *We have heard that their wings are great and decided it was finally time to check it out. Their wings are whole wings and crispy, which is a nice change of pace.*

Its "bag" of unigrams extracted from the sentence is:

```
['We', 'have', 'heard', 'that', 'their', 'are', 'great', 'decided',
'finally', 'time', 'check', 'out', 'Their', 'are', 'whole', 'which',
'nice', 'change']
```

Due to the simplicity of unigram features, a MaxEnt model with only unigrams as features is trained as a baseline model. A significant but easy improvement on this model is to treat a single gram as not only a tokenized word but as the pair containing the word *and* its POS tag, distinguishing the different semantic meanings a word may embody.

5.4 Negation Detection

Phrasal negation is an important syntactic feature that is almost impossible to detect with a simple bag-of-words approach. Consider the following sentence from the data set:

> *The dips weren't anything super special.*

Both the words *super* and *special* suggest positive polarity. However, the existence of the word *weren't*, or more specifically the possessive ending *n't*, signals a reverse in polarity. One approach, experimented in [2], is to hand select a set of negation words and negate the sentiment of a number of the words that appear after a negation word. A good scope of negation, or the number of words to apply negation to, falls around 3, 4 or 5 [2].

One way to model this is to prepend a *'NOT_'* prefix to all words that falls within the scope of negation and treat them as new features [10]:

The dips weren't NOT_anything NOT_super NOT_special.

The negation words tht we will be considering are *"no"*, *"not"*, *"rather"*, *"hardly"*, and verbs ending in *"n't'*, all of which were tested in [2].

5.5 SentiWordNet

SentiWordNet[4] is a lexical resource for opinion mining [1] that is based on WordNet [9]. WordNet contains a large collection of sets of cognitive synonyms that express a distinct concept called *synsets*. SentiWordNet builds upon WordNet by assigning each synset of WordNet three sentiment scores - positivity, objectivity and negativity - such that for any synset, or *sentisynset*, in SentiWordNet

$$positivity(\text{sentisynset}) + objectivity(\text{sentisynset}) + negativity(\text{sentisynset}) = 1 \tag{10}$$

In this scoring system, it is possible for a word to have both a positive sentiment and a negative sentiment. But the sentiment of any word lies on a spectrum from 0 to 1, with 0 being the most negative and 1 being the most positive.

We will use SentiWordNet through NLTK. For example, consider the word *recommend*, which has a sentiment score highly skewed to *objectivity* when used in context as a verb[5].

```
>>> from nltk.corpus import sentiwordnet as swn
>>> swn.senti_synsets('recommend')
[SentiSynset('recommend.v.01'), SentiSynset('commend.v.04'),
SentiSynset('recommend.v.03')]
>>> recommend = swn.senti_synsets('recommend.v.01')
>>> recommend.pos_score(), recommend.obj_score(), recommend.neg_score()
(0.0, 0.875, 0.125)
```

SentiWordNet inherits its POS tags from WordNet, which uses a very different tag set from the Penn Treebank tagset. It has only four POS, one for each of *noun, verb, adverb, adjective*. For each pair of word and POS tag, there are an indefinite number of word sense, each belonging to a synset. While we can manually map a Penn Treebank POS tag to its counterpart in WordNet's POS tagset, there is unfortunately no interface for WordNet to pick the most suitable word sense when given a Penn Treebank POS tag. To escape the problem of word sense disambiguation, we will consider only the first word sense, or word sense '01'.

5.6 Base Syntactic Chunks

Base syntactic chunks are the syntactic features of a sentence at the base level, or patterns of POS. As with bag-of-words, we extract syntactic paths that have a significant tf-idf score. The size of the patterns, or the number of grams of POS tags that make up a feature, can be anything larger than 1.

Another variation of this is to use constituent syntactic paths. However, due to time constraints, parsing the datasets to obtain the parse trees of sentences would take too long and therefore was not achievable.

[4] http://sentiwordnet.isti.cnr.it/

[5] For more SentiWordNet usage in NLTK, see http://www.nltk.org/howto/sentiwordnet.html

6 Results and Discussions

An overview of experimental results is shown below (Table 2). A more comprehensive review of the performance of each model, including confusion matrices, can be found in Appendix B. All of the following models can be found in `models.tar.gz`.

#	Features	F-Measures (%) Pos	Obj	Neg	Accy. %
1	Unigrams	88.6	0.0	53.7	68.8
2	Unigrams with POS	78.0	0.0	52.2	65.0
3	Unigrams with POS and Negation	77.9	0.0	52.6	65.0
4	Unigrams with POS and Negation*	83.3	0.0	55.0	67.6
5	Unigrams with POS, Negation and Syntactic Chunks*	82.9	0.0	54.9	67.5
6	SWN Valued Unigrams with POS and Negation	80.9	83.9	83.9	88.7

*Trained as a binary model instead of trinary.

Table 2: Overview of Selected Experimental Results on Development Set

Baseline Model Model 1, which uses only unigrams as features, was trained as a baseline Model. In this model, no POS tags or SentiWordNet scores were used to help extract a bag-of-words. Rather, words are chosen purely according to their tf-idf scores. This worked surprisingly well, especially with classifying positive reviews.

Unigrams v ngrams Many of the aforementioned models were repeated on bigrams and even trigrams. However, they performed either as good as or worse than unigram models. While ngrams do capture more contextual information than does unigrams, they do not effectively capture the information that adds value to sentiment analysis [5].

Part-of-Speech There are two ways POS tags are used in these models. First, it serves to help identify a word's synset in SentiWordNet, which contains the positivity, objectivity, and negativity scores of a word. Second, the base syntactic chunks of POS can be used as features alone. This is implemented in the module `SyntacticChunksFeatExtractor.py`[6]. The first usage of POS tags is instrumental to ngram bag-of-words extraction, whereas the second usage, which was incorporated in model 5, reflected poorly on the development set. A similar result was obtained in [5], which also showed that POS patterns alone do not improve performance.

Negation Negation detection works by identifying negation words and prepending a prefix to words that follow the negation word. This way, the words with this new prefix, or *negated words*, are treated as new features. A large proportion of the most informative features[7] of these MaxEnt models are negated features (Table 3), which goes to show the level of importance of negated features.

[6]See `FeatExtractor.py`

[7]Obtained by querying `show_most_informative_features` in NLTK's `MaxentClassfier`

	Weights	Feature	Label
1	7.811	(u'NOT_unassuming', u'JJ')	Negative
2	7.599	(u'NOT_medium', u'JJ')	Positive
3	6.126	(u'NOT_inconvenient', u'JJ')	Positive
4	6.015	(u'NOT_unassuming', u'JJ')	Objective
5	5.595	(u'NOT_naturally', u'RB')	Positive

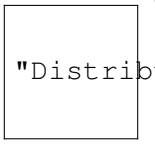
Table 3: The five most informative features of model 3

Binary v Trinary A consistent problem that runs across most tested models is their inability to detect objective reviews. All objective reviews are wrongly classified as negative. To combat this problem, a different approach to the problem is to build a binary MaxEnt classifier using only positively and negatively labeled training data (see 5.2). In classifying new reviews, the MaxEnt classifier will find the probability that a review is positive or negative. If the probability of being positive and the probability of being negative is relatively equal, then the review will be classified as objective. Otherwise, it will be classified it as whichever one is more probable.

The results, however, were much less ideal. All objectively labeled reviews were still wrongly classified as negative. In practice, it seems intuitive that probability distribution should be uniform. That is, if the difference between the probabilities of being positive and negative for a review falls within 33% of each other, then that review is considered objective. Given that we trained a binary classifier, let p be the probability that a review is positive. Then the probability that a review is negative is $1 - p$. Therefore, we classify reviews with $p \geq 0.67$ as positive, $0.33 < p < 0.67$ as objective, and $p < 0.33$ as negative. In reality, a substantial amount of reviews with $p \approx 0.108$ are objective (Figure 1). The probability distribution is heavily skewed (Figure 2), which resulted in the failure in detecting objective reviews.

Figure 1: Predicted Label v Actual Label for Model 4

Figure 2: Predicted v Actual Probability Distribution for Model 4



SentiWordNet SentiWordNet is an invaluable resource to this classification problem. There are two ways SentiWordNet is used in this project, both of which serves to help to extract a bag-of-words. First, in the case of binary classification, it identifies words that have a nonzero positive or negative sentiment. This helps filter out words that are entirely objective, which does not add value to the binary classification problem that involves strictly classifying a review as either positive or negative. This type of feature extraction is implemented in the module `FeatExtractor.BoWFeatExtractor`[8] and used in models 2 to 5.

Our second approach at utilizing SentiWordNet is to include any unigrams in the training set that has a sentiment score skewed to the classification it appeared in. We then proceed to

[8]See `FeatExtractor.py`.

identify the most significant label associated with any unigram. Therefore, the vocabulary of unigrams for which we will consider as features consist of the most significantly correlated classification label *in addition* to the unigrams themselves. In extracting features, the feature value is no longer 1 for occurrence and 0 otherwise, but rather the SentiWordNet score of the unigram and 0 otherwise. This type of feature extraction is implemented in the module `FeatExtractor.SWNBoWFeatExtractor`[9] and is used in model 6.

As we can see, the most accurate model here is model 6, which is by far the only model that can correctly identify objective reviews. This is the model that was eventually used on the test set, which yielded an average accuracy of 80.2%.

	Positive	**Objective**	**Negative**
Recalls	80.4	56.2	74.1
Precisions	72.8	63.0	73.9
F-measures	76.4	59.4	74.0
Accuracies	83.5	74.4	82.7

Table 4: Performance of Model 6 on Test Set

7 Conclusion

In this sentiment analysis project, we trained a Maximum Entropy classifier to classify a Yelp Review as positive, objective, or negative. We tried several different combinations of features, including bag of word, syntactic chunks and explicit negation detection. We also utilized outside resources such as Stanford POS tagger and SentiWordNet to extract meaningful features from the text. Overall, a model trained on a training set of 40 000 reviews was able to achieve 80% accuracy on the test set. From this model we can see that the some of the most informative features to sentiment analysis lies in the individual *sentiment* each word carry, and not so much in the syntactic features of the text. A bag-of-words approach, therefore, is most suitable for sentiment analysis.

References

[1] Andrea Esuli Baccianella, Stefano and Fabrizio Sebastiani. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining.

[2] Maral Dadvar, Claudia Hauff, and FMG De Jong. Scope of negation detection in sentiment analysis. 2011.

[3] James H Martin and Daniel Jurafsky. Speech and language processing. volume 710, chapter Hidden Markov and Maximum Entropy Models. 2000.

[4] Bruno Ohana and Brendan Tierney. Sentiment classification of reviews using sentiwordnet. In *9th. IT & T Conference*, page 13, 2009.

[5] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[9] See `FeatExtractor.py`.

[6] Henrique Siqueira and Flavia Barros. A feature extraction process for sentiment analysis of opinions on services. In *Proceedings of International Workshop on Web and Text Intelligence*, 2010.

[7] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[8] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003.

[9] Princeton University. About WordNet. wordnet. http://wordnet.princeton.edu, 2010. Accessed: 2016-12-15.

[10] Michael Wiegand, Alexandra Balahur, Benjamin Roth, Dietrich Klakow, and Andrés Montoyo. A survey on the role of negation in sentiment analysis. In *Proceedings of the workshop on negation and speculation in natural language processing*, pages 60–68. Association for Computational Linguistics, 2010.

Appendices

A Usage Documentation

class `FeatExtractor`**`.BoWFeatExtractor`** *(vocab = None, saved_vocab = None, negate = True)*

Base: `FeatExtractor`

A bag-of-words feature extractor that can build training toks compatible with NLTK's `MaxentClassifier` and extract bag-of-words feature vectors from a text review. Initialized with either a set of features `(vocab)` or a JSON file of vocab previously saved.

Typically, this is created with the `classmethod train()` from a collection of training sets.

class `FeatExtractor`**`.SWNBoWFeatExtractor`** *(vocab = None, saved_vocab = None, negate = True)*

Base: `FeatExtractor`

A SentiWordNet valued bag-of-words feature extractor that can build training toks compatible with NLTK's `MaxentClassifier` and extract SentiWordNet valued bag-of-words feature vectors from a text review. Initialized with either a set of features `(vocab)` or a JSON file of vocab previously saved.

Typically, this is created with the `classmethod train()` from a collection of training sets.

class `FeatExtractor`**`.SyntacticChunksFeatExtractor`** *(vocab = None, saved_vocab = None, negate = True)*

Base: `FeatExtractor`

A syntactic chunks feature extractor that can build training toks compatible with NLTK's

`MaxentClassifier` and extract bag-of-words feature vectors from a text review. Initialized with either a set of features `(vocab)` or a JSON file of vocab previously saved.

Typically, this is created with the `classmethod train()` from a collection of training sets.

class **`SentimentClassfier`** *(feature_extractors, model = None, saved_mode = None)*

Base: `object`

A sentiment classifier that is built upon NLTK's `MaxentClassifier`. Initialize it with a list of feature extractors (`FeatExtractor`) and either a `MaxentClassifier` model or a saved `pickle` file of an `MaxentClassifier` model. Once initialized, it can be used to evalute the performance of the model on other datasets using `evaluate()`.

Typically, this is created with the `classmethod train()` from a set of feature extractors (`FeatExtractor`) and a collection of training sets.

Below is an example usage of `SentimentClassifier` using default paramters.

```
>>> from FeatExtractor import BoWFeatExtractor, SNWBoWFeatExtractor,
SyntacticChunksFeatExtractor
>>> from SentimentClassifier import SentimentClassifier
>>> training_sets = ["training_set_tagged_pos.json",
...                  "training_set_tagged_obj.json",
...                  "training_set_tagged_neg.json"]
>>> test_sets = ["test_set_tagged_pos.json",
...              "test_set_tagged_obj.json",
...              "test_set_tagged_neg.json"]
>>> fvec_extractor1 = BoWFeatExtractor.train(training_sets)
>>> fvec_extractor2 = SNWBoWFeatExtractor.train(training_sets)
>>> fvec_extractor3 = SyntacticChunksFeatExtractor.train(training_sets)
>>> fvec_extractors = [fvec_extractor1, fvec_extractor2, fvec_extractor3]
>>> classifier = SentimentClassifier.train(fvec_extractors, training_sets)
>>> classifier.evaluate(test_sets)
```

B Selected Experimental Results

All of the following models were trained for 100 iterations on the training sets with an Improved Iterative Scaling algorithm (IIS) via NLTK's `MaxentClassifier`. Some experiments that displayed highly uninformative results are omitted. All data shown are obtained from validating the model on the development set.

B.1 Unigrams

`>>> model1_unigrams.pickle`

A baseline trinary class model was trained with only unigrams as features. This model performed moderately well with positively labeled reviews (89% F-measure) and decently well with negatively labeled reviews (54% F-measure), but did not perform well at all with objective reviews. In fact, it incorrectly identifies all objective reviews as negative. The high training accuracy (92%) but low Development accuracy (69%) suggest that this model highly overfits the training data. Bigram-only and trigram-only versions of this same model were trained but performed much worse than unigrams – both yielding only 55% accuracy.

Actual \ Pred.	Pos	Obj	Neg
Positive	1418	202	46
Objective	0	0	1665
Negative	117	311	1238

Table 5: Confusion matrix for model 1

	Pos	Obj	Neg
Recalls	0.851	0.000	0.743
Precisions	0.924	0.000	0.420
F-measures	0.886	0.000	0.537
Accuracies	0.927	0.564	0.572

Table 6: Evaluation metrics for model 1

Training accuracy: 0.922 Development accuracy: 0.688 Trinary or Binary: Trinary

B.2 Unigrams with POS tags

```
>>> model2_unigramsPOS.pickle
```

This trinary model was also trained on unigrams, but the individual unigrams of interest are pairs of a word and its POS tag. Hence, distinctions can be made among words that have multiple meanings. To some degree, however, this model performed worse than the model with unigrams without POS tags (B.1) – F-measures and accuracies are brought slightly down. This model is still remarkably insufficient at detecting objectively labeled reviews.

Actual \ Pred.	Pos	Obj	Neg
Positive	1139	370	157
Objective	0	0	1665
Negative	116	318	1232

Table 7: Confusion matrix for model 2

	Pos	Obj	Neg
Recalls	0.684	0.000	0.739
Precisions	0.908	0.000	0.403
F-measures	0.780	0.000	0.522
Accuracies	0.871	0.529	0.549

Table 8: Evaluation metrics for model 2

Training accuracy: 0.698 Development accuracy: 0.650 Trinary or Binary: Trinary

B.3 Unigrams with POS tags and Negation

```
>>> model3_unigramsPOS_negation.pickle
```

In this model, we introduced negation detection. When a negation word is detected, the words within negation scope following a negation word are prepended with a special prefix, or *negated*. An interesting finding is that the most informative features are all negated features, which shows that these features significantly impact the outcomes.

Actual \ Pred.	Pos	Obj	Neg
Positive	1142	395	129
Objective	0	0	1665
Negative	123	308	1235

Table 9: Confusion matrix for model 3

	Pos	Obj	Neg
Recalls	0.685	0.000	0.741
Precisions	0.903	0.000	0.408
F-measures	0.779	0.000	0.526
Accuracies	0.871	0.526	0.555

Table 10: Evaluation metrics for model 3

Training accuracy: 0.721 Development accuracy: 0.650 Trinary or Binary: Trinary

B.4 Binary Model Using Unigrams with POS tags and Negation

```
>>> model4_unigramsPOS_negation_binary.pickle
```

This is a binary MaxEnt classifier using only positively and negatively labeled training data (see 5.2). In classifying new reviews, the MaxEnt classifier will find the probability that a review is positive or negative. If the probability of being positive and the probability of being negative is relatively equal, then the review will be classified as objective. Otherwise, it will be classified it as whichever one is more probable.

Actual \ Pred.	Pos	Obj	Neg
Positive	1255	280	131
Objective	0	0	1665
Negative	94	260	1312

Table 11: Confusion matrix for model 4

	Pos	Obj	Neg
Recalls	0.753	0.000	0.788
Precisions	0.930	0.000	0.422
F-measures	0.833	0.000	0.550
Accuracies	0.899	0.559	0.570

Table 12: Evaluation metrics for model 4

Training accuracy: 0.902 Development accuracy: 0.676 Trinary or Binary: Binary

However, all objectively labeled reviews were still wrongly classified as negative. A investigation into the problem shows that the probability distribution heavily skewed instead of uniform, which resulted in failure in detecting objective reviews.

B.5 Binary Model using Unigrams with POS tags, Negation and Syntactic Chunks

```
>>> model5_unigramsPOS_negation_synChunks.pickle
```

In this model, we introduced base syntactic chunks as features. Base syntactic chunks are three consecutive POS tags in any sentence. These newly added features performed made almost no difference to the performance of the model, as accuracy is still leveled at 68%. Similar models were trained with chunks of size 2 and 4, but they resulted in no or negligible improvement.

Actual \ Pred.	Pos	Obj	Neg
Positive	1246	285	135
Objective	0	0	1665
Negative	94	260	1312

Table 13: Confusion matrix for model 5

	Pos	Obj	Neg
Recalls	0.748	0.000	0.788
Precisions	0.930	0.000	0.422
F-measures	0.829	0.000	0.549
Accuracies	0.897	0.558	0.569

Table 14: Evaluation metrics for model 5

Training accuracy: 0.707 Development accuracy: 0.675 Trinary or Binary: Binary

B.6 SentiWordNet Valued Unigrams with POS tags and Negation

```
>>> model6_unigramsPOS_negation_swn.pickle
```

In this model, we further utilize SentiWordNet as a resource by extracting unigram feature values from it. Concretely, we identify the classification label that a unigram is most significantly correlated with in the training set, and set the value of the features to be the SentiWordNet sentiment score of the combination of the word, POS tag, and most likely label.

This approach considerably improved the performance of the model, raising accuracy by 30% from a stable 67% to 88%. In addition, this is the first time the model has been able to correctly detect *any* objective reviews. A similar binary model was trained using only positive and negative training data, but that resulted in a decline in accuracies from 88% to 78%.

Actual \ Pred.	Pos	Obj	Neg
Positive	1198	357	111
Objective	0	1665	0
Negative	98	284	1284

Table 15: Confusion matrix for model 6

	Pos	Obj	Neg
Recalls	0.719	1.000	0.771
Precisions	0.924	0.722	0.920
F-measures	0.809	0.839	0.839
Accuracies	0.887	0.872	0.901

Table 16: Evaluation metrics for model 6

Training accuracy: 0.794 Development accuracy: 0.887 Trinary or Binary: Trinary