Sequence Tagging and Sentiment Classification Report

Mateo Acebedo, Teng Qu, Karl-Frederik Maurrasse

1. Introduction

In this project, we implement a classifier based on Hidden Markov Model (HMM). With the pre-labeled training data we compute transition probabilities from one sentiment to another; and with some self-constructed features we compute the emission probabilities of each sentence being in a certain sentiment. Then using Viterbi algorithm, we compute the sequence of sentiments that will maximize the path's probability.

2. Model Description

- **Notations:** We use '-1' to denote negative sentiment, '0' to denote neutral sentiment and '1' to denote positive sentiment. In calculating transition probabilities, we also define a start symbol 's' so that the first sentence can be taken into account (more on this in the Transition Probabilities section). In emission probability, we use {epos, eneu, eneg} to denote the probabilities that a sentence is positive, neutral and negative respectively.
- **Transition Probabilities:** These probabilities are defined to be the probability from one state to another. We decided to first try bigram transition probabilities, which means the probability for the following sentence to be in a state is only dependent on its immediate precedent. Specifically, in this project, the transition probabilities are the probability from -1 to -1, -1 to 0, -1 to 1, 0 to -1, 0 to 0, 0 to 1, 1 to -1, 1 to 0 and 1 to 1. We also need the probabilities of the starting sentences: s to -1, s to 0, s to 1. These probabilities are calculated by counting how many times $state_1$ follows $state_2$ and divide this count by the total occurrences of $state_2$ (notice here we do not count sentences crossing reviews, i.e. the first sentence of $review_2$ is not following the last sentence of $review_1$).
  Later in the **Extensions** section, we will describe an experiment with trigram based transition probabilities, which uses the previous two sentences to predict the following sentence's state probabilities.
- **Feature Vector:** A feature vector is a representation of a given sentence on the feature space we defined. It is a string of -1's, 0's and 1's, whose components are all calculated by individual feature functions which we will describe later in the **Feature Description** section.
- **Emission Probabilities**: These probabilities are the probabilities of a sentence being in a certain state given its presented features. Specifically, we want to compute a probability for each possible feature vector. For example, assume we have 2 features, each taking values of either '1' or '0'. Then the task is to compute a mapping from '11', '10', '01' and '00' to a set of probabilities which reflect given this feature vector, how probable is the sentence in positive state, neutral state or negative state. These probabilities are computed by counting the number of positive/neutral/negative sentences having a certain feature vector and dividing it by the total number of sentences having such feature vector.

- **Viterbi Algorithm:** We use the Viterbi algorithm described in class to decide a sequence of sentiment tagging that maximizes the path's probability. Specifically, if a review contains n sentences s_1, s_2, …, s_n, for each sentence s_i, we first compute its emission probability {$epos_i$, $eneg_i$, $eneu_i$}, then multiply $epos_i$ by the transition probabilities from positive, negative and neutral to positive and perform the similar calculation for $eneg_i$ and $eneu_i$. In each step, we record the sentiment that will maximize the probability till that step.

After performing the Viterbi Algorithm, we get a sequence tagging for a given sentence.

3. Data Preprocessing

We have also incorporated some pre-processing steps, such as dictionaries of positive and negative words[1], as well as part-of-speech tagging.
In the following description, we put our source code function names in the square brackets.

- **Part-Of-Speech Tagging** [strip]
  This pre-processing attempts to reduce a given sentence to a compact form by eliminating parts of speech[2] that are usually not determining factors in sentiment classification. Our intuition here is twofold: 1) a shorter sentence will carry less noise (particularly with certain features we have considered below); 2) considering key parts of a sentence allows us to hone in (hopefully) on the determining factors of sentiment classification (adjectives, adverbs, etc).
- **Smoothing** [Good-Turing]
  We incorporated Good-Turing smoothing to account for unknown observations.

4. Feature Description

We designed and tested a number of features for this project. Below we are describing each of them and report their performances. Since features are independent from each other, we run the experiments independently first, and then explore their performances when applied together. Good features should be features yielding extremely high or low probabilities on one sentiment; bad features are those which split the probabilities evenly among positive, neutral and negative.
In the independent evaluation step, we compute the emission probability and get result in the format of a python dictionary. The keys are 1 if the sentence has such feature and 0 otherwise; and values are quadruples (positive probability, neutral probability, negative probability, number of sentences exhibiting such feature).

1) **Sentence Length** [isLong][3]

[1] List of positive and negative words are extracted from
http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#lexicon
[2] Part of speech tagging is acquired from NLTK library
[3] We define sentences longer than 42 words to be long sentences. 42 is selected to make the probabilities skew to one sentiment based on experiments.

Neutral sentences (and reviews for that matter) are expected to be longer than their positive or negative counterparts. Our intuition is that people will list positive and negatives aspects of a topic, causing the sentence to run.
Computing emission probability on sentence length alone gives us the following result:
{'1': (0.31, 0.5, 0.19, 256.0), '0': (0.26, 0.41, 0.33, 1776.0)}.
This feature is effective to identify which sentences are not likely to be negative ones as long sentences have only 19% chance to have negative sentiment. However, since there are only few sentences exhibiting this feature, it is not a great feature.

2) **Punctuation** [containsExclamation]
We also believe that an exclamation point can be indicative for non neutral sentences.
The emission probability of this feature is shown below:
{'1': (0.37435897435897436, 0.2717948717948718, 0.35384615384615387, 195.0), '0': (0.23477493380406, 0.4333627537511033, 0.3318623124448367, 2266.0)}
The result shows that not too many neutral sentences have this feature, which is what we have expected. The problem for this feature is too small sample size is supporting this feature, making its application less popular.

3) **Capitalization** [containsAllCaps]
Here, our intuition was that a word in all uppercase characters typically conveys a strong sentiment from the author.
The emission probability with this feature is shown below:
{'1': (0.22, 0.43, 0.35, 970.0), '0': (0.26, 0.42, 0.32, 1491.0)}.
The result shows that few (22%) sentences exhibiting this feature are categorized as positive. However, in the sentences that do not exhibit this feature, the results are about the same. That means this feature does not work the way we have hoped. A reason for this is that a lot of abbreviations also involves all capitalized words and their presence will distort this feature.

4) **Negation** [containNegation]:
We constructed a negation word list[4] and test if a sentence contains any negation words. The intuition here is that if people use negation words, they are probably trying to express non-positive feelings. It is anti-intuitive to use negation words in positive feeling expressions.
The result is shown below:
{'1': (0.1676190476190476, 0.4247619047619048, 0.4076190476190476, 525.0), '0': (0.2959522229595222, 0.4173855341738553, 0.28666224286662245, 1507.0)}
The result shows that when there is at least one negation word in the sentence it is not likely to be positive. With only 16% sentences exhibiting this feature are expressing positive sentiments, this feature can be used to identify non-positive sentences from positive ones.

5) **Sentence Polarity** [getPolarity]
A more involved feature, this getPolarity function attempts to guess at the correct sentiment of a sentence by tallying up its positive negative, and neutral words and

[4] Negation word list constructed from http://www.grammarly.com/handbook/sentences/negatives/

classifying the sentence based on what type has the majority within the sentence. Essentially, a majority vote on sentiment. We have also tried variants of this feature (one variant only tallies positive and negative words and classifies as neutral only if those counts are equal), but they appeared less promising than a straightforward majority vote. {'1': (0.43, 0.38, 0.19, 822.0), '0': (0.18, 0.5, 0.32, 640.0), '-1': (0.12, 0.38, 0.5, 570.0)}. This feature is by far the best feature we have. The results show that when a sentence is composed of mainly positive words, the sentiment of this sentence is not likely to be negative and more likely to be positive, and the same thing for the other two sentiments. Moreover, large number of sentences for each category are identified, making this result highly reliable.

6) **Baseline Model**
   We use feature 5 alone to build the baseline classifier and present the result below.
7) **Aggregated Results**
   After performing independent tests, we run some experiments on different ways to aggregate features to form a feature vector and generate emission probabilities based on feature vector. The impact of aggregating different features are listed below:

Features used	Accuracy on Cross Validation	Accuracy on Testing
1) through 5) with strip	50%	43%
1) through 5)	44%	42%
1) through 4) with strip	44%	43%
1) through 4)	41%	42%
Baseline (Use only 5))	44%	43%

5. Cross Validation

Given the fact that the training data is fairly small (we have only about 190 reviews and about 2000 sentences for training), we decided to run k-fold cross validation on the training data set. This approach holds (1/k)th data for validation and train on the rest data, then use the reserved validation set to configure the model parameters. Run the training-validation step for k times on k different validation sets and average the accuracy across these k validations will give us a relatively more accurate validation result.

6. Extensions

We implemented 2 extensions in order to improve our prediction accuracies.

- **Trigram assumption:**
  As an extension we decided to calculate transition probabilities using a trigram model as opposed to the bigram model. Our hypothesis was that the sentiment of a given

sentence would not only depend on the previous step, but also on the one before the previous. We believed that using this trigram model would improve our overall sentiment analysis algorithm. We calculated these transition probabilities by using sum($state_x$ follows $state_y$, $state_z$)/sum($state_y$, $state_z$) where x,y,x belong to {positive, neutral, negative}. We further combined this extension with the extension "Incorporated review sentiments" described below in order to compare accurately the trigram assumption versus the bigram assumption. The results are discussed in the **Results and Discussion** section.

- **Incorporate review sentiments:**
  Intuition: We perform this extension because during validations on the initial model, we noticed that a lot of sentences are labeled as '0'. This can be a result of transition probabilities from start to neutral is too high, making the first sentence label is always neutral. Once the first sentence is labeled as neutral, the transition probability from neutral to neutral is over 50%, under which circumstance requires extremely high positive or negative emission probabilities to label the following sentences non-neutral. We assume reviews with different sentiments will start differently and therefore use this extension to combat this problem.
  In this extension we splitted the training data into 3 subsets by review sentiments. Then we compute the transition probabilities for each subset. In the prediction step, we first check what sentiment the review has. If the sentiment of the review is positive, we use the transition probabilities trained from positive reviews, and perform similar steps for neutral and negative reviews.

7. Results and Discussions

The final model gives us 50% accuracy in 5-fold cross validation and 43% accuracy on Kaggle. The difference between cross validation and testing data is noteworthy. One reason for this is the limited size of the training data. Without large enough training data, the model could be a local optimum which performs poorly in certain cases.

We also noticed that in our model a sentence is seldomly predicted to be positive. This is because: (i) we do not have particularly good features to identify the positive sentences; (ii) the transition probability from start to positive is not taking majority, making the first sentence rarely positive. As we carefully reviewed all transition probabilities, it is very common that following sentiment s_1, the most probable successor is also s_1, i.e. if the previous sentence is neutral, then the probability the next sentence being neutral is fairly high. Since the probability of a sentence being positive is defined to be the transition probability from previous sentiment to positive (low due to (ii)) and the emission probability of the sentence being positive (low due to (i)), it will be unlikely that the model predict many positive sentences.

We found it quite surprising that our bigram assumption led to better results than the trigram assumptions. We believe this happens because the training data is quite small and thus a more information demanding model, like the trigram model, ends up performing poorly. It would be

interesting to see if our hypothesis that the trigram probability model for the transition probabilities does in fact improve when the training data size increases.

A careful look into the training data makes us believe the training data itself is not perfect. It tends to give more emphasis on neutral. For example, we discovered the following labeled sentence in the training data:

[neu] I'm sorry people no one can beat or sing Commissioned songs better than Commissioned!!

This sentence, in our opinion, should be categorized as positive instead of neutral. There are many neutral sentences in the training data that have this problem. We used to consider the double exclamation mark as an indication of non-neutral sentiments. However, the emission probability computed for this feature is close to 0.3 : 0.4 : 0.3, which means this feature cannot classify anything.

Here is a screenshot of our performance (We are team NoBias):

26	new	(ノ °□°)ノ ⌒ ┻━┻	0.54412	22	Wed, 30 Apr 2014 21:11:03
27	↓10	MRVL	0.51961	10	Wed, 30 Apr 2014 03:27:33 (-0.3h)
28	new	daniel2kobe	0.50490	16	Thu, 01 May 2014 01:27:02 (-2.2h)
29	↓14	MarseillesBiggestFan	0.50163	46	Thu, 01 May 2014 01:51:03 (-31.5h)
30	↓10	VokramNeddih	0.48366	132	Wed, 30 Apr 2014 01:40:10 (-7.5h)
31	↓9	abms	0.48203	158	Thu, 01 May 2014 01:22:05 (-7.7h)
32	↓14	Team Name Goes Here	0.46242	48	Wed, 30 Apr 2014 18:18:03 (-2.9h)
33	new	exhaustedStudents	0.45261	1	Wed, 30 Apr 2014 01:02:20
34	↓13	domdom	0.44608	3	Mon, 28 Apr 2014 20:27:53 (-0.1h)
35	new	1st place	0.44444	9	Thu, 01 May 2014 01:46:33 (-0.8h)
36	new	**NoBias**	**0.43464**	**9**	Thu, 01 May 2014 01:51:02 (-37.5h)

Future Work

As we have pointed out in **Results and Discussion**, the system is not good at identifying positive sentences. This can be improved by: 1) discover more features that are good at identifying positive sentences; 2) incorporate more sentence level features such as parse trees; 3) if possible, acquire more data to train the system.

Individual Member Contribution

Mateo: Feature testing, trigram model implementation and testing, report

Karl: Feature testing, model implementation and testing, report

Teng: Feature testing, cross validation, model construction and testing, report

References

1. Bird, Steven, Edward Loper and Ewan Klein (2009), *Natural Language Processing with Python*. O'Reilly Media Inc.

2. http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#lexicon

3. http://www.grammarly.com/handbook/sentences/negatives/