User Review Rating Prediction

Sammy Leong, Ashish Bhatia

Dec 14, 2012

Objective

We're given a set of user review data each of which is associated with a user rating. Our objective is to build a model that is able to predict whether an unseen review is positive or negative.

Overview

To solve this problem[4, 3], first we evaluated a number of learning algorithms such as Logistic Regression, Naive Bayes, and SVM on the original raw data as-is to get a sense of their feasibility, capability, and speed. Then we developed our own pre-processing pipeline to transform the original data to better emphasize sentimental features, which we evaluated with one of the learning algorithms.

Based on our analysis, we removed several redundant and noisy features to further improve the accuracy of our pipeline.

Data Source

We gathered our user review data from the Yelp academic data set[1] which contains user reviews of local businesses near Stanford University.

Each review contains a text sequence representing the user review along with a numerical rating that ranges from 1 to 5. The first thing we did was relabel each review as positive or negative. Reviews with ratings $1, 2$ are labeled as negative, reviews with ratings $4, 5$ are labeled as positive, and reviews with rating 3 are randomly labeled as positive or negative.

As it turns out, 80% of the reviews are positive which means the data is highly skewed to begin with. To address this, we reshuffled the data such that we have equal number of positive and negative reviews, and that positive and negative reviews are interleaved. We had to throw out a lot of positive reviews but we're still left with abundant training examples.

Success Metric

Our goal is to find a learning algorithm along with data processing techniques that give us the highest generalization accuracy. To do this, we divided our data in half, trained with the first half, and then tested with the second half. The former gives us the training accuracy while the latter gives us the generalization accuracy.

First Attempt (Baseline)

For our first attempt we focused on trying out various learning algorithms on the original unprocessed data as-is to get a sense of their capability for our given problem. Another goal is to choose the fastest algorithm which gives reasonably good results and use it to iterate while we work on our pre-processing pipeline.

Results

Here is a table that shows the training and generalization accuracies for each learning algorithm, along with the time it took to run. The experiments were done using 14000 training examples and 14000 testing examples.

	Train	Test	Time
Matlab			
Naive Bayes (mn)	91.67	81.27	1.05
Liblinear			
Logistic Regression	97.39	81.67	1.58
L2-reg SVM (linear)	99.69	78.71	2.86
Libsvm			
C-SVM (linear)	98.99	78.86	786
C-SVM (radial)	70.26	69.68	356
C-SVM (sigmoid)	65.11	65.43	359
nu-SVM* (linear)	87.78	83.75	277
nu-SVM* (radial)	88.80	83.91	296
nu-SVM* (sigmoid)	84.78	81.49	291

* $nu = 0.5$

Discussion

Firstly, it's quite clear that liblinear runs significantly faster than libsvm. For this reason, we will iterate using liblinear when we evaluate the performance of various pre-processing techniques. In particular, we will use logistic regression because it has higher generalization accuracy than L2-regularized SVM with linear kernel.

Secondly, it appears that nu-SVM with $nu = 0.5$ in this case improved both the training and generalization accuracies significantly (as compared to the counterpart C-SVM results). In fact, the we were able to achieve the highest generalization accuracy using nu-SVM with the radial basis kernel. The only issue is that it takes a very long time to run. For this reason we will only revisit it after we've settled on a good preprocessing pipeline.

Second Attempt

For our second attempt, we focused on data processing and feature selection with the goal of reducing the noise in the data as much as possible. Here we only made use of logistic regression with 14000 training examples and 14000 testing examples. What follows are the data processing techniques that we used.

Spell Correction

The first thing we did was to apply spell checking on the reviews and replace mispelled words with suggested corrections. The idea here is that user reviews on the internet are often filled with misspelled words. One positive review may contain the word "good" while another may contain "gooodi". We want to treat both signals as representing positive reviews.

Stemming

After spell correction, we further consolidated words to their canonical forms by applying stemming. For example, one positive review may contain the word "perfect" while another may contain "perfection". Again, we want to treat both signals as representing positive reviews. Stemming converts both words to their root: "perfect".

Stopword Removal

In this step we removed stop words that are believed to add little relevence to reviews. Words like "the", "I", "was", etc. are stop words and thus removed. One caveat is that we did not remove the words "no" and "not" which will be explained in the bigram generation section that follows.

Bi-gram Generation

After spell correction, stemming, and stopword removal, we moved on to bi-gram generation. This step is very important because it allows us to capture semantics that are not captured in the uni-grams or worst have opposite meaning altogether. For example "really good" is a much stronger signal for positive reviews than "pretty good" or just "good". Similarly, "not good", if captured inidividually contains the word "good" which falsely indicates a positive review. We can fix that by adding the bigram "not-good" but now we have a problem where the review contains contradictory signals. To fix that, we remove the word "good" and keep only "not-good" which is a clear indication of a negative review.

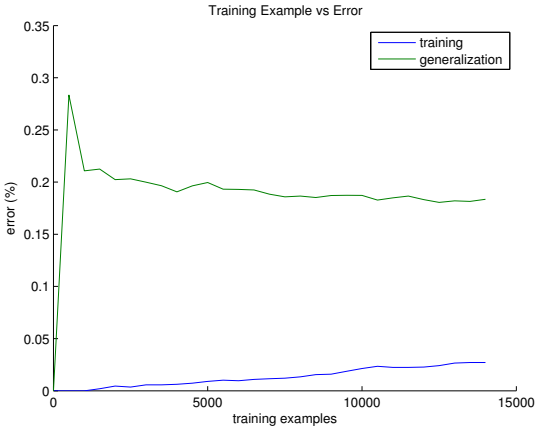
Results

Here is a table that shows the training and generalization accuracies. The *Individual* section shows experiment results where we used each data processing technique individually. The *Incremental* section shows experiment results where we incrementally combined data processing techniques. Finally, the *Optimal* section shows experiment results where we combined

only data processing techniques that we believe to give us optimal results.

	Train	Test
Individual		
Baseline	97.3931	81.6759
Spell Correction	97.1862	82.3517
Stemming	95.5655	82.1586
Stopword Rem	97.1724	82.6759
Bi-grams	99.9241	84.4621
Incremental		
Baseline	97.3931	81.6759
+ Spell Correction	97.1862	82.3517
+ Stemming	95.5034	82.0000
+ Stopword Rem	95.3241	81.5586
+ Bi-grams	99.7379	83.2966
Optimal		
Spell + Bi-grams	99.9586	84.5103

Here is a graph that shows the number of training examples vs training/generalization error for the case of using only spell correction and bi-grams.



Discussion

Individually, every data processing technique improved the generalization accuracy but all of them with the exception of bi-grams, also reduced training accuracy. When combined incrementally, however, it appears that stemming and stopword removal actually reduces generalization accuracy.

From the individual and incremental experiment results, we gathered that the most promising data processing techniques are spell correction and bi-grams. Thus we combined only those two and indeed we achieved training and generalization accuracies that were superior to the rest.

Looking at the training error vs generalization error, it is as expected that by adding more training examples, the training error (bias) goes up where as the generalization error (variance) goes down. However the trend suggests that as we add more training examples, training error will continue to go up where as the generalization error will likely flat off. Judging from this we will have to either explore other pre-processing techniques, other learning algorithms, or other tuning techniques.

Further enhancements

To further refine our results, we focused on data processing and feature selection with the goal of reducing redundant features in the data. Just like in second attempt, we only made use of logistic regression with 14000 training examples and 14000 testing examples. What follows are the extra data processing techniques that we tried.

Using tri-grams (and beyond)

We tried using tri-grams, quad-grams and even penta-grams. The assumption here was that the it will catch phrases like "not so good", "was hardly any better" but it turns out that these phrases have really sparse density. Therefore, we did not get any significant gains out of it. In some trials, the accuracy even took a dip.

Respecting natural phrase boundaries for bi-grams

Earlier, we were forming bigrams across boundaries like full stop, question marks and semicolons, this produced features where the first word was a part of previous phrase and second part of another. During the analysis of misclassified results, we realized that this was creating a lot of noise and hence, we added checks to ensure that we do create bi-grams across phrase boundaries.

Importance of parts of speech

A detailed analysis of results showed that proper nouns like "pizza", "pasta" as well as determiners like "which" were too frequent and therefore, ended up in features. To clean this up, we decided to consider only following parts of speech.

This cleanup, though, made the feature extraction phase really slow, gave us signficant boost in accuracy. We used NLTK[2] for parts-of-speech tagging.

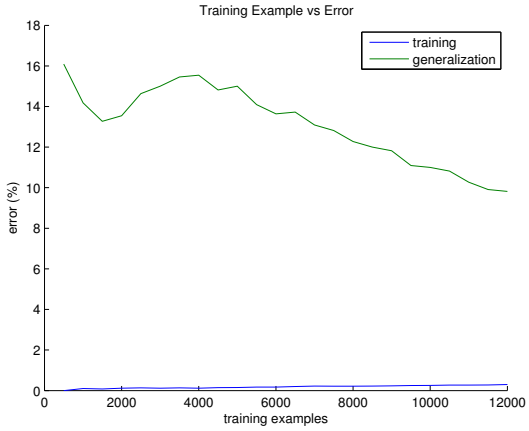
POS Tag	Description	Example
CC	Conjunction	And
CD	Cardinal number	1, 2
EX	existential there	there is
JJ	adjective	green
JJR	adj. comparative	greener
JJS	adj., superlative	greenest
RB	adverb	good
RBR	adverb, comparative	better
RBS	adverb, superlative	best
VB	verb, base	take
VBD	verb, past tense	took
VBG	verb, present participle	taking
VBN	verb, past participle	taken
VBP	verb, singular present	take
VBZ	verb, 3-rd person	takes
WP	wh-pronoun	who
WP$	wh-possessive	whose
WRB	wh-adverb	when

Results

Here is a table that shows the training and generalization accuracies. The *Individual* section shows experiment results where we used each data processing technique individually. The *Incremental* section shows experiment results where we incrementally combined data processing techniques. Finally, the *Optimal* section shows experiment results where we combined only data processing techniques that we believe to give us optimal results.

	Train	Test
Individual		
Baseline(spell + bi-grams)	99.9586	84.5103
Tri-grams	99.9602	84.6311
Quad-grams	99.7413	83.4371
Phase boundaries	99.9754	85.6752
POS selection	99.9723	90.1685
Incremental		
Baseline (spell + Bi-grams)	99.9586	84.5103
+ Tri-grams	99.9602	84.6311
+ Quad-grams	99.7731	83.4513
+ Phase boundaries (pb)	99.9013	85.1846
+ POS selection	99.8979	89.5943
Optimal		
Baseline + Bi-grams + pb + POS	99.9679	90.3476

Here is a graph that shows the number of training examples vs training/generalization error for the case of spell correction + bi-grams + phase-boundaries + POS (parts-of-speech) selection.



Discussion

Clearly, using k-grams for k>2 is too noisy. Ignoring bi-grams across phrase boundaries (like full stop) improved the results and using Natural language tool kit to tag parts of speech and then removing extraneous parts of speech gave a huge boost to accuracy of our pipeline.

References

[1] Yelp's Academic Dataset. On request, 2012.

[2] Steven Bird. Nltk: The natural language toolkit, 2002.

[3] Shoushan Li, Sophia Y. Lee, Ying Chen, Chu-Ren Huang, and Guodong Zhou. Sentiment Classification and Polarity Shifting. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 635–643, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[4] Lizhen Qu, Georgiana Ifrim, and Gerhard Weikum. The bag-of-opinions method for review rating prediction from sparse text patterns. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 913–921, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.