Sentiment Classification on Steam Reviews

Rohan Bais, Pasal Odek, Seyla Ou

I. Introduction

Sentiment Analysis is a machine-learning topic that has been performed frequently and exhausted quickly in many scenarios, given the high demand to understanding sentiment within a body of any general text. However, language is not always laid out in such a straightforward fashion. In real world examples, people do not simply echo or spam highly polarizing positive and negative words to make it easy for machine learning tactics to classify; human-written text is much more nuanced than simply that. In day-to-day life, text has personality written into it: text can have a mixture of conflicting feelings that may make it hard to classify into one definite category, text can have cultural references or memes embedded into them that make it easy for humans but hard for computers to spot the sentiment, and most troublesome of all, text can have sarcasm imbued over it that can sometimes even trick humans into thinking theyre genuine reactions. Since real-world text is more subtle and nuanced than a simple positive and negative binary classification, we wanted to test the bounds of binary sentiment classification with positive and negative labels, using a more hip and modern-lingo based review system as a resource, so we turned to the Steam gaming platform's review system.

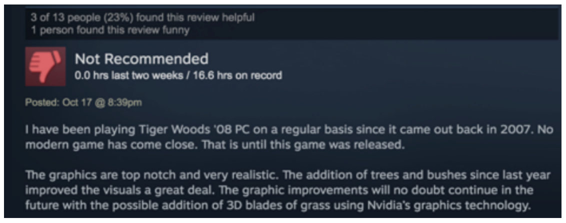
Steam is a multi-billion dollar distributed gaming platform that acts as a third-party medium to sell games online and download them. Generally, the user-base for this platform has been young (it ranges from 18 to 30), so naturally, because it is a game-review platform and the base is somewhat young, the culture around Steam is built on sarcasm, memes, and wit.

With trickier texts such as Steam game reviews, we hoped to build a binary sentiment classifier to peg reviews as positive and negative, seeing if it would work with the language subtleties explained earlier. The input to the sentiment classifier is a plain-text review attached with the metadata surrounding the review such as percent of people who found the review helpful, funny, and number of hours the reviewer played. Using these inputs, we used the approaches of SVMs, Logistic Regression, Multinomial Naive Bayes, Turney's unsupervised phrase-labeling algorithm, and a lexicon-based baseline.

II. Related Work

Despite the problem not being a completely novel one, various people have been working on sentiment classification to see if it can overcome language subtleties like mentioned earlier and achieve perfect accuracy, not even missing out sarcastic reviews. Pang and Lee tried sentiment classification of IMDB reviews but tried varying the feature set to include unigrams, bigrams, and part-of-speech tagging, to improve the generic unigram method as the go-to feature set. The results were promising, as bigrams kept track of some context of the words that the bag-of-words model did not, although the part-of-speech tagging provided no valuable increase in accuracy[1]. Another approach built off of the approach here by Pak and Paroubek, and used a more generic n-grams model, with n up to 3, to try and fully capture the context of the situation and included some pivotal features such as emoticons and used these for the SVM and Naive Bayes approaches. Emoticons, like ":)" and ":(" are generally strong indicators of sentiment as they are brief and very rarely do they matter so much in contexts like individual words do. With these changes, they improved upon Pang and Lee's work [2]. Other methods, such as the models from Prusa and Dittman, rely more on the unigram model directly but add weighting schemes such as TF-IDF (term-frequency and inverse-document-frequency) and hand-writing a couple of rules with additional weighting when words are italicized or bolded [4]. The results have also been proven to do better than the standard unigram model, as unigrams just weigh on a basis of term frequency without consideration of uniqueness of a particular word and how pivotal it is. It better achieved accuracy than the previous two approaches, but still failed on classifying sarcastic reviews.

Others people, such as Turney, tried to approach this in a different regard with unsupervised learning and phrasal detection. Turney used POS-tagging to extract phrases and determined a semantic orientation by calculating co-occurrence with a phrase and a positive and with the phrase and a negative word (e.g. "excellent" and "poor") [3]. Then he averages the scores of all phrases in a review to see the sentiment. This has proven somewhat successful as his extraction gets key phrases out and ignores the fat and not useful text in a lot of reviews, but it lacks context sometimes and classifies incorrectly when mistaking a description of a movie as an actual sentiment. State-of-the-art is geared more towards recursive or convolutional neural networks since they eliminate high-bias and high variance. A paper from Ren and Zhang shows that recursive neural networks combined with word2vec word embeddings, which turns words into vectors that are close if the words are similar and far apart if not, achieves a very high accuracy, with the only flaws being its inability to detect sarcasm [5]. Our approach shares much in common with these mentioned approaches despite the difference in datasets, we use Turney's algorithm and we rely not only on words and word-weighting as features, but we also use features outside of the individual words in the review, like hours played or percent found review helpful, to better achieve accuracy during sentiment classification and avoid the same pitfalls of not detecting sarcasm without context.



3 of 13 people (23%) found this review helpful
1 person found this review funny

Not Recommended
0.0 hrs last two weeks / 16.6 hrs on record

Posted: Oct 17 @ 8:39pm

I have been playing Tiger Woods '08 PC on a regular basis since it came out back in 2007. No modern game has come close. That is until this game was released.

The graphics are top notch and very realistic. The addition of trees and bushes since last year improved the visuals a great deal. The graphic improvements will no doubt continue in the future with the possible addition of 3D blades of grass using Nvidia's graphics technology.

Fig. 1. An example of a sarcastic review that is negative but appears positive. It is hinted to be sarcastic because it highly praises an obscure game and uses hyperboles(Tiger Woods '08)

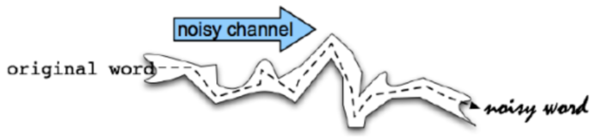


Fig. 2. The noisy-channel represents spelling errors like substitution, deletions, and insertions of extraneous letters

III. Dataset and Features

A. Data Collection and Preprocessing

There was not an explicit dataset for Steam game reviews, as the idea of classifying game reviews is more novel than classifying movie reviews, and any dataset that is available only gives the literal text rather than any interesting features surrounding the review. The review in figure 1 is an example of the review and how it looks on a website. Using the words alone would make it somewhat difficult to truly tell that this review is negative, so we gathered the dataset manually to fully capture this wealth of information.

We wrote some sample Javascript code that identifies these review HTML elements on the website, loops through each one of them up to a certain amount, extracts the relevant features and texts, and writes it to a text file in JSON format. We ran this code with a limit of 50 review extractions per page for every positive and negative review filtered game page for 100 different games to retrieve 5000 total labeled examples, which we later used for a 70-30 10-fold cross validation split. The games we chose have some similar yet different data distributions: there are games with massive followings with widespread critical acclaim (Undertale, Super Meat Boy), games with a smaller following that are less critically acclaimed (Dustforce, Hotline Miami), and games that are large-based with flat-out negative reception (Call of Duty, No Mans Sky)

Preprocessing varied depending on the supervised or unsupervised learning algorithm we chose. For all methods, we included an NLTK noisy-channel spelling corrector model, depicted in figure 2, that checks the conditional probability of a word given the misspelled word, and if high enough, around high 90 percent, we correct the word. NLTK finds these probabilities internally through comparison of deletion, insertion and substitution errors, the computation of Levehnstein distances, and seeing if the corrected word makes sense in context of the sentence. We also remove stop-words, words such as "the" and "you", that do not really indicate anything about sentiment and only bloat the text. Additional preprocessing included part-of-speech tagging, so that Turney's algorithm could extract relevant phrases. For other algorithms, we utilized the Porter Stemmer to truncate suffixes as any past, present, future tense of a word should really be the same feature since they all mean the same thing.

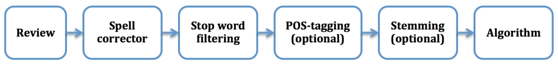


Fig. 3. A pipeline representing the preprocessing of the review data. The last two steps were optional depending on the algorithm used

B. Features

1) TF-IDF with positional weighting: Generally for many sentiment classification methods, the individual words are themselves used as features, but we thought that this might not be enough. Even with stop words filtered out, there are common words that will be included in the feature vector that will have the same weighting as more indicative and unique words. With this shortcoming, we decided to use the tf-idf weighting scheme which weighs words heavily that occur frequently in a specific document and less frequently in other documents and lessens that weight as that word appears in more documents. The positional weighting was something we thought we could do to improve the weighting since we hypothesized that sentiment tend to aggregate towards the beginning and end of the document and the middle is more for game description. Here was the exact equation we used:

$$weight_{w,r} = \left(tf_{w,r} \times log\left(\frac{N}{df_{word,N}} \right) \right) \times p(r,w)$$
$$w = word, r = review$$

The inner most expression represents the TF times the IDF of the word and the left expression, $p(r,w) = 0.5cos(2\pi x) + 1.5$ is a weighting scheme that takes the position of a word and maps it to a 0 to 1 scale, which our positional weighting scheme uses and outputs a scaling factor between 1 and 2 to scale the tf-idf by. The tf-idf scheme helps emphasize the more important words to a document and thus gives more weight to those indicative words to allow specific features to be better used in classification.

2) Percent found review helpful: It is important to understand the setting of a game community forum to help classify and understand the sentiments around the reviews. Game communities have massive followings and because of this, people more or less spam likes or dislikes on favorable and unfavorable reviews respectively, no matter the validity of the sentiment or well-constructed the criticism. In the dataset, people with negative reviews generally had low "found helpful" ratings whereas people with positive reviews mostly had high "found helpful" ratings. This was more because of the massive fan-base of the game liking and disliking positive and negative reviews respectively, so it could be used as a distinguishing feature.

3) Percent found review funny: Funny and humorous reviews are something that can be tied to both positive and negative reviews (in our dataset, positive funny reviews tended to be more self- deprecatory about "crying while playing an emotional game" whereas negative reviews lambast with ridiculous analogies and lengthy rants). The same echo-chamber phenomenon described earlier appears here as well: positive reviews were found more funny and negative reviewers hit more soft spots and were generally found less funny, even if the negative review was somewhat humorous.

4) Number of hours played: The number of hours played also tends to show a trend for positive and negative reviews. Positive reviews will generally have the users playing upwards from 20 to 1000 hours (depending on how replay-able the game is). Negative reviews, on the other hand, will have fewer than 10 hours played, mostly due to the fact that if someone found a review negative, he or she wouldnt want to spend too much time playing and reliving negative experiences, which keeps the hour count small.

IV. Methods

A. Lexicon Score Aggregation

Our baseline consisted of just simply utilizing a dictionary of positive and negative words and simply just aggregating the total counts and checking which . We thought to try this as this was a simple method to implement and we wanted to see where exactly it would fail on subtleties in language and how it would fare relative to the other methods above this baseline.

B. Multinomial Naive Bayes

For the next method, we thought we would try Naive Bayes, since that is the next step up or is usually a baseline. It was used with the features being a "bag-of-words" (only counts of unigrams, no order particularly) approach combined with stop word filtering. It breaks down a review using the unigrams as features and after training and computing conditional probabilities of words appearing in positive and negative reviews, it uses these probabilities directly to classify any test data

$$c_{predicted} = argmax_c P(r|c)P(c)$$

$$c_{predicted} = argmax_c \left(P(c) \times \prod_{w_i \in r} P(w_i|c) \right)$$

It tries to find the probability of a certain class given the review $P(c|r)$ and uses Bayes Rule to get the first equation. We are interested in the class that maximizes the probability, and the second equation's product comes from the independence assumption that the features (words) are independent of each other and can be split into a product of probabilities. We tried this method out to see the shortcomings of Naive Bayes and to see if the independence assumption or other types of sentences trips the classifier up.

C. Modified Turney's Algorithm

We wanted to see how an semi-supervised learning algorithm would compare to the more supervised-heavy algorithms that we would use later on. Turney's algorithm works by extracting key phrases. Phrases follow a certain set of rules countable by hand. A phrase consists of two words where: the first is an adjective and the second is a noun, the two are both adjectives, the first is an noun followed the second as an adjective, or the first is a adverb and second is a adjective. As an example the first pattern can be exemplified with the phrase "great game" (adjective followed by noun) and the fourth pattern can be exemplified with the phrase "absolutely terrifying" (adverb followed by noun).

Given the reviews, in a training and test split, it takes the reviews of the training data, extracts all of the phrases out of these reviews and the first 10 words surrounding the phrases. It also keeps track of the all positive and negative words in the training data reviews using the same dictionary in the lexicon method as a set of top words. When classifying a review in the test scenario, it extracts all the phrases in the review, checks the training set if that phrase appeared (if not ignore it) and then it calculates the point-wise mutual information (PMI) between that phrase and all the positive and negative words found in the set of top words and computes the semantic orientation by taking all of the pointwise mutual information of positive words and subtracting them by pointwise mutual information of negative words. Using the semantic orientation of a phrase, it aggregates the semantic orientation of all of the review's phrases and uses that number to determine if a review is positive or negative (positive average semantic orientation is positive, negative is negative). Here are the equations:

$$PMI(p, w) = \frac{\#\ of\ times\ p\ appears\ with\ w}{\#\ of\ times\ w\ appears}$$

$$SO(p, r) = \sum_{w \in r_+} PMI(p, w) - \sum_{w \in r_-} PMI(p, w)$$

The first equation just simply counts co occurrence by checking how many times a word appears with a phrase and dividing it by how many times a word appears. The second equation, describing the semantic orientation, works by calculating by summing all of the PMIs in r_+, which is all positive words in a review r, and summing all the PMIs in r_-, which is all negative words in r and subtracting the two. It differs from Turney's original slightly in that Turney's only used the words "excellent" and "poor" to check semantic orientation, which we felt like wasn't enough. Thus we generalized it to work with all positive and negative words in reviews, though it becomes more expensive doing so.

D. Logistic Regression

Logistic Regression is a Generalized Linear Model that is used widely for multi-class classification. It works by trying to fit the sigmoid function, $g = \frac{1}{1+e^{-x}}$, to the individual labeled points (0 for negative, 1 for positive in a binary classification example). It has a hypothesis function $h(x) = g(\theta^T x)$ where x is the feature vector of an example and θ is a weighting vector learned, which represents the probability that x is classified as positive (similarly, $1 - h(x)$ is the probability x is negative). It primarily trains itself using these data points by using a stochastic gradient ascent update rule:

$$\theta_j := \theta_j + \alpha(y^i - h(x))x^i_j$$

where α is the learning rate to affect how quickly the program converges and how it steps. After the weights are given, we simply compute the hypothesis function.

E. Linear SVM

Linear SVM is a method that creates a classifier (a vector) that separates the two labeled point, at least in the binary classification case. Geometrically given the two types of points, circles and x's, in a space, it tries to maximize the minimum distance from one of the points to the place. In other words, it maximizes the margin. Although kernel tricks can be applied to make the data more linearly separable, we did not think it was necessary as we have high-dimensional feature vectors that are most of the time linearly separable. The optimization problem the SVM tries to solve is below:

$$min_{\gamma,w,b}\frac{1}{2}||w||^2$$

$$s.t. y^i(w^T x + b) \geq 1, i = 1, 2, ...m$$

It tries to find the w to satisfy the maximum margin problem and satisfy the separability constraint.

In addition to using all of these methods, we wanted to test our feature set using the same method to judge how the features fared with classification capability, so we used Linear SVM for this task as well as comparing the algorithm's classification capability to other methods mentioned earlier.

V. Experiments and Results

Since we are performing binary sentiment classification, we chose accuracy, precision and F1-score as metrics of performance, as well as the original test and training set accuracies.

Our experiments mainly delved into two types: one experiment was to test how the algorithms themselves fare against each other over an increasing training set size. The methods to compare include the baseline, Naive Bayes, Logistic Regression, and Turney's Algorithm. The second set of experiments was to check what exactly was the ideal feature set and how feature selection affected the same metrics (precision, accuracy and recall). For both of these experiments, we performed 10-fold cross validation with a 70-30 percent training-test data split as we wanted to see how a moving window of training and test sizes affect the metrics and see whether or not that tells us about the distribution and similarity/differences amongst the data points. The training-test split was approximately around 3500 reviews in the training set and 1500 reviews in the test set for each of the 10 folds

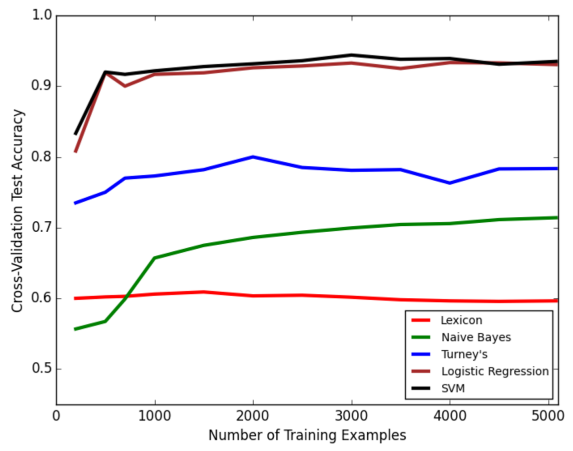


Fig. 4. Average accuracy over # of examples

Results of various algorithms average metrics

Method	**Train Accuracy**	**Test Accuracy**	**Precision**	**Recall**	**F1-score**
Baseline	0.596	0.596	0.569	0.79	0.661
Naïve Bayes	0.868	0.714	0.669	0.847	0.748
Turney's	0.792	0.731	0.782	0.742	0.764
Linear SVM	0.939	0.935	0.909	0.99	0.947
Logistic Regression	0.935	0.931	0.91	0.975	0.944

Fig. 5. The average metrics for each of the 5 methods

For our first experiment, testing the various methods against each other, we performed the cross-validation and reported the average results in figure 4. In figure 5 we show how the algorithms perform over a variable number of learning examples, starting from 200 going all the way to 5000 examples.

Our baseline, as expected under-performed, with an average accuracy of 59 percent. The errors for this method sprouted when it couldn't find the subtleties in the sentences. For many positive examples, there were reviews saturated with positive words and fewer negative words but it was not able to grasp the concluding sentiment, despite being much shorter than the rest of the review, is the true sentiment. As an example, "I thought the prequel was fantastic and great, but this is horrendous" starts off positively and ends negatively, but our method would see more positive words than negative words and label it positive.

Naive Bayes was also seemed to have fared better as it performed better over more training examples. This method; however, suffered many of the short comings the baseline had. Again, because of the independence assumption, the words' order is lost. Texts can have the same or nearly the same bag of word feature set and still have opposite meanings. It doesn't fare well against the earlier example as well, as it will find fantastic and great to have high probabilities for a positive label, but it should have really focused more on the statement after the "but" transition. Looking at the graph and the table, it had a higher variance and this probably was because with just the bag of words feature, the training and test data split

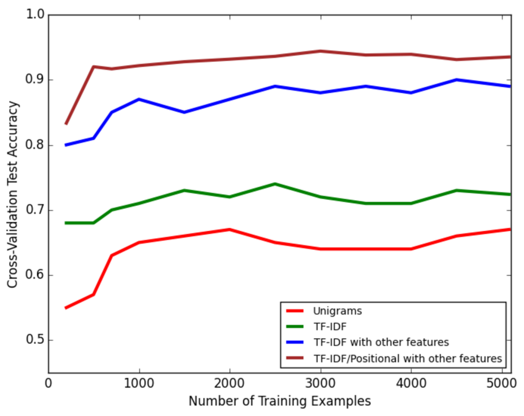


Fig. 6. Average accuracy over # of examples, with SVM

Predicted (Unigrams)

Actual	Pos.	Neg.
Pos.	0.65	0.35
Neg.	0.24	0.76

Predicted (TF-IDF)

Actual	Pos.	Neg.
Pos.	0.74	0.26
Neg.	0.16	0.84

Pred (TF-IDF + other)

Actual	Pos.	Neg.
Pos.	0.87	0.13
Neg.	0.15	0.85

Pred (Positional TF-IDF + other)

Actual	Pos.	Neg.
Pos.	0.87	0.13
Neg.	0.15	0.85

Fig. 7. Confusion matrices associated with each feature set

didn't come from the same data distribution.

Turney's algorithm; however, made some better use of context of sentences. Turney's algorithm only focuses on a couple of key phrases and in sentiment detection, these key phrases are really the points of interest. Even with stop-word filtering there is a lot of fat in the text that needs to be cut. It also uses PMI to determine semantic orientation and this has to keep track of the 10 nearest words around a phrase, which already preserves more of the order than the previous two methods did. This could be a good explanation of why it does slightly better than Naive Bayes, and it didn't fail on the same example the other two methods did. However, it did fail on some reviews that needed more context. Some phrases the algorithm would extract would be relevant to the game but not to the review itself. In other words, it mistook description for review, like when describing the plot of a game and I mention "evil wizard", it will include it as a negative sentiment when I really did not portray my feelings at all. It seemed not to really suffer from high variance as Naive Bayes did.

Next we tried out the Logistic Regression model. This model has a very close performance to the SVM, as we can see from the graph, as it initially does worse and perhaps has a little more bias, but then smooths out to the same level. These minor differences would probably have to do with the fact that we regularized it, which makes it so it doesn't overfit in the training set. However, this seemed to have similar results, and the training and test error are very close as expected from regularizing the regression. However, even with its additional features, it still suffered from the lack of context and preservation of order around the sentences and it did so when the bag-of-words wasn't indicative enough and the number of hours played and percent found helpful features were not helpful to classify.

Support Vector Machines performed interestingly well. Compared to others, it had the highest metric scores so in that regard it performed well. However, looking at the context of the features and the SVM applied with different feature set, we can tell what was really beneficial. The unigram feature set did worse than the tf-idf over increasing example size, which proves the fact that unigrams equally weighing non-indicative words did in fact hurt the accuracy. The tf-idf scheme by itself also did considerably worse than tf-idf with the additional features like (hours played, etc.) and this was most likely due to the fact that tf-idf failed on sarcastic reviews. It failed on sarcastic reviews, but when including features like number of hours played and percent found funny, it is able to see the true sentiment (sarcastic reviews that seem bad verbally would have a huge playing time). Lastly, the positional weighting also did help out a lot, as I was concerned that even tf-idf would put too much focus in the middle of the text when the true sentiment is at the fringe of the reviews. It did perform better, but this assumption made it fail in other places, like reviews where the sentiment is purely in the middle and the beginning and end are more logistic game descriptions.

VI. Conclusion

Language will always remain subtle, nothing, like sentiment, will purely remain black and white. This subtlety is what made us passionate to pursue this project to see if machine learning concepts can capture it properly. We tried two experiments: one where we tried out various methods like Naive Bayes and Turney's, and another method where we used SVM but varied our feature set to determine the best combinations. From observing, SVM with TF-IDF and positional weighting outperformed the others because with additional features like hours played and proper weighting of indicative words, it is able to work through sarcastic reviews and determine the true sentiment. Future work would include trying out neural networks, both convolutional neural nets and recursive, to see if it really does eliminate high bias and high variance in this scenario. In addition to this, we laid out the groundwork for sarcasm detection, so possible future work could be seeing if sarcasm detection directly is possible, given these features and additional ones.

VII. References

[1] Lee, Lillian, et al. "Thumbs up? Sentiment Classification Using Machine Learning Techniques." 2001 Cornell CS Journal.

[2] Pak, Alexander et al. "Twitter as a Corpus for Sentiment Analysis and Opinion Mining" January 2010, DBLP Journal

[3] Prusa, Joseph et al. "Impact of Feature Selection Techniques for Tweet Sentiment Classification" May 2013, Florida Aritifical Intelligence Society Conference

[4] Turney, Peter "Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews" July 2002, Association for Computational Linguistics

[5] Ren, Yafeng, Zhang, Yue "Context-Sensitive Twitter Sentiment Classification Using Neural Network" January 11, 2016 AAAI publication

VIII. Contributions

Rohan: Implemented tf-idf weighting scheme, as well as cross validation and the various SVM methods and Turney's. Helped write up the final write-up.

Seyla: Implemented the Naive Bayes method with cross-validation as well and helped with the poster and write-up mostly. Proposed and determined feature sets for testing as well.

Pascal: Implemented the baseline and wrote the html parsing code to extract each of the 50 positive and 50 negative reviews from the 100 games we chose. Also helped write this up.