NLP & Machine Learning Applied: Video Game Reviews

Matthew D. Mulholland
Montclair State University
Montclair, NJ
mulhollandm2@montclair.edu

Abstract

Despite the ambiguity of the concept, much research has been done in the area of detecting "fake" reviews. It is, however, often difficult to build corpora containing reviews that are definitively "fake" or "true". It is simpler to reframe the issue in terms of the amount of experience a reviewer has with a product: given a review, can we tell anything about the level of experience the reviewer has with the reviewed product?

In this exploratory paper, I will detail a research project whose aim was to relate video game reviews to proxies for reviewer experience, such as number of hours played, number of times marked as helpful, and other related review/user attributes, using natural language processing and machine learning techniques. The ultimate end is to produce a capability for ranking or filtering reviews, which could be used in addition to or in place of other fake review or spam filtering algorithms.

A less grand aim of the project was to scrape review data from the Steam video game website and make it publicly available. This data will be described at length.

1 Credits

I would like to thank both Janette Martinez and Emily Olshefski for their help in the initial iteration of this work as part of a class project. They helped lay out the problem, make decisions regarding the source of the data and the games for which data was collected, and also write some of the preprocessing code. Some sections of this paper build on the final paper for that class project, which they took a lead in writing.

2 Introduction

2.1 Reframing a familiar problem

In the realm of deception detection, ground truth – information that can be verified or denied – is not easy to come by. For example, one application of deception detection is in the detection of fake reviews: reviews that are known to be fake (by some external means) are compared to reviews that are believed to have been written in good faith. It may be easy in some cases to identify certain reviews as fake since the authors themselves might admit as much. However, the categorization of other reviews from either category is a difficult matter. Thus, corpora of fake/real reviews are often constrained in that, for any given review, it could be impossible to determine the category.

In this paper, I propose a fundamental reframing of the issue: the problem should not be about whether or not a given review is fake, but rather about measuring the amount of experience a reviewer has with the product being reviewed. Reviews for products of which the reviewer has little to no experience – whether they have been written in bad faith or simply from a relatively uninformed perspective – could be distinguished from reviews for which the reviewer does have experience with the product being reviewed. Further, different levels of experience – say, moderate or high – could be distinguished from one another.

A system that could accurately predict the amount of experience a reviewer has, given only the review text or some combination of the review text and other attributes of the review/reviewer, such as the number of times a review has been marked as helpful, could be useful in a production environment as a component in a review-filtering and/or -sorting algorithm.

2.2 How is experience measured?

To model reviewer experience (which is, after all, just as nebulous as the distinction between fake and true reviews), there must be methods of approximating experience. One simple way to approximate this value is by recording the amount of time a user has actually spent using a product before reviewing it. For most products, however, the feasibility of recording the actual time spent is low due to the cost and logistics of such an undertaking. And that is assuming time even can be recorded! In other cases, product use is more of a binary value: Did the consumer eat the pie or not? Did the user watch the movie or not? And, in still other cases, usage might need to be measured in terms of the number of times the product was used, such as lotion, hair spray, etc.

Aside from trying to measure a reviewer's actual usage of a product, there may exist other ways of approximating experience. One rather indirect way would be to compare the number of times each review is marked as helpful (or not helpful). Presumably, reviews that are more helpful are reviews that are from users who actually used the product and vice versa.

2.3 Using video game reviews from Steam

In the realm of video games, the methods of approximating user experience described above are actually feasible. The Steam online video game platform (`http://store.steampowered.com/`) is used by video game enthusiasts for a variety of purposes. Steam functions as a platform

- in which users can play video games in an online, social environment
- from which users can purchase, research, and review video games
- for social interaction (through posting and commenting on video game reviews, marking reviews as helpful, funny, etc., displaying playing stats and achievements, etc.)

Steam keeps a record of the number of hours each user has played each game and, thus, when a user submits a review of a video game, this information is presented alongside the review. It is true that the meaning of this measure is not completely straightforward: a user could have played a particular game for many hours outside the Steam platform and these hours would not be included. However, it is a reasonable assumption to make that the majority of the values recorded by the platform will be accurate and/or that the amount of time that players have played a particular game outside the confines of the online platform will be similar across players submitting reviews for the same game.

The window into a user's experience with a product that is afforded by the amount of time a user has used a product is somewhat unique to the case of video games: for other products, it might be impossible or pointless to attempt to record the amount of time the user used the product. For example, there is no way to keep a record of the amount of time a user has spent using a vacuum short of conducting a study and painstakingly recording such information.

However, if experience could successfully be modelled in the case of video games, perhaps the models could be generalized to cover whole categories of products and, thus, the situation here would apply to much more than video games. Furthermore, there are other indications of a user's experience with a product, as mentioned above, such as the number of times a review has been marked as helpful, and fortunately Steam also keeps a record of such information.

2.4 Description of research

In this paper, I explore the amount of time a reviewer spent playing the game he/she is reviewing and the number of times it was marked helpful by review readers in relation to review text itself (and also to a set of attributes about the review/reviewer, such as the reviewer's number of friends, the number of times the review was marked as funny, etc.). The greater the amount of experience a reviewer has with the game being reviewed, the greater the likelihood is that the review produced is trustworthy or a relatively true depiction of product use. Naturally, if a reviewer has spent a lot of time playing a game or if the review is voted to be relatively helpful, etc., then the review has a higher chance of representing a good understanding of the game on the part of the reviewer.

Fortunately, the online gaming platform Steam collects such data about its users and makes it publicly accessible. A web-scraping method was developed and used to build a corpus of review texts along with a lot of additional data, as mentioned briefly above. Reviews from 11 of the most popular games were scraped from the Steam website. After filtering out non-English reviews, the data was partitioned into training/test sets from which samples could be culled in a repeatable fashion and without possible contamination. A number of commonly-used NLP feature types were extracted from the reviews and machine learning experiments were conducted using a range of learning algorithms.

The main motivation of the machine learning experiments was to determine whether or not the various proxies for experience under consideration demonstrated potential in terms of whether or not they could successfully be modelled. The results are mixed, but there is some indication that this is a worthy research effort and that it could lead down the road to some of the grand aims mentioned above, i.e., influencing a review filtering/sorting algorithm.

3 Related Work

While this paper does not fall under the purview of typical deception detection work, influences from other deception detection research form the basis for some of the underlying ideas of this research. (Jindal and Liu, 2008) used product review data from `Amazon.com` to find "opinion spam". Discovering opinion spam led to research into deceptive reviews. Since there was no gold standard data to work from at that time, they initiated detection of such spam by first detecting duplicate reviews and then by using supervised learning with manually-labeled training examples. They gathered reviews from different categories such as music, books, and DVDs, but video games were not specifically included. Amazon reviews were also used by (Fornaciari and Poesio, 2014) to test identification of fake reviews using crowdsourcing.

(Ott et al., 2011) developed gold standard deception review data consisting of 400 true and 400 false reviews. The 400 fake reviews were written by Amazon Mechanical Turkers while the real ones were mined from `Trip Advisor.com` (and pruned to match the criteria given to the turkers). Naive Bayes, Support Vector Machines (SVM LIGHT), and the Stanford Parser were used and an accuracy of 89.6% was achieved using bigram features. (Feng et al., 2012a), using Ott's gold-standard corpus, extracted PCFG parse trees for deep syntax encoding along with some shallower syntactic/semantic features (such as LIWC features and POS tags to garner comparisons of effectiveness) in addition to the original bigram features. With support vector machine, they were able to improve on Ott's accuracy score, increasing it to 91.2%.

One advantage of using data from Steam (as opposed to `Amazon.com` or `TripAdvisor.com`, both of which have stored tranasctional data) is that Steam's data goes a step beyond the ground truth of the aforementioned websites. Not only is transactional data easily accessible, but the fact is that hours played values provide a window to the ground truth of product experience. There is no practical litmus test for the prior works to test trustworthiness in the same way. Rather, prior works have had to resort to circuitous ways of testing the trustworthiness of reviews.

4 Data and Code

4.1 Data

Data from 11 popular video games was scraped from the Steam website via a web scraping program written in Python. See Table 1 for a full listing of the games and the number of reviews that were collected for each game (after filtering of non-English reviews, reviews that had zero content, etc.). The program was designed to scrape the following (non-exhaustive) types of information from reviews:

- review text
- amount of time reviewer spent playing game (in hours)
- number of times a review was marked helpful
- number of times a review was marked unhelpful
- number of times a review was marked funny
- number of comments other users have left in the review's discussion space
- number of friends the reviewer has on the Steam platform
- number of Steam groups the reviewer is a member of
- number of reviews, guides, and screenshots the reviewer has posted (for this and other games)
- number of Steam "badges" the reviewer has received
- number of in-game achievements the reviewer has attained

All data-sets have been made freely available with a permissive license (MIT) via `GitHub.com`. The data can be found at the following URL: `https://github.com/mulhod/steam_reviews`.
It is stored in relatively common and portable `JSONLINES/NDJ` format. The data was collected over a period of approximately two weeks in the summer of 2015.

Game	# reviews
Arma 3	7,151
Counter Strike	6,040
Counter Strike: Global Offensive	7,073
Dota 2	9,720
Football Manager 2015	1,522
Garry's Mod	7,151
Grand Theft Auto V	13,349
Sid Meier's Civilization 5	7,467
Team Fortress 2	5,676
The Elder Scrolls V	7,165
Warframe	7,123

Table 1: Review data-sets.

4.2 Code

Like the data, all code written for and used in this project has been made freely available via `GitHub.com`. The code repository can be found at the following URL: `https://github.com/mulhod/reviewer_experience_prediction`. Also provided are in-depth descriptions of the data, full ReST-style documentation for the code, clear instructions on how to get set up and running, and numerous Jupyter (formerly known as IPython) notebooks describing various algorithms and aspects of the code and providing visualizations of the data, etc. All code is written in the Python programming language and the total number of lines of code exceeds 6,000. Cython is used to convert large parts of the code base to C extensions to improve performance. The `spaCy` Python library is used for most of the natural language processing (NLP) analysis.

5 Methods

After collecting reviews from the Steam website, the reviews were analyzed and a set of NLP features were extracted. All NLP features and review/reviewer attributes were stored in a MongoDB database collection. MongoDB is particularly useful in cases where the data is unstructured: it allows for easily inserting new fields for existing document entries. In the case of this project, the amount of data and the time it takes to extract NLP features for each review text necessitated some creative thinking in terms of how best to store the data and reduce the computational load/the time needed to conduct many different experiments with potentially the same data. It was determined that a database holding all reviews and all NLP features would be the best solution despite the fact that the database would need a lot of memory devoted to it – the database ended up needing over 200 GiB of hard-drive space! The benefits of having a running database are that the data can be accessed at any time and that it can be queried in a way that would be impossible if it existed in flat files.

5.1 NLP feature extraction

Using the `spaCy` Python library for NLP analysis and other tools, all review texts were analyzed and NLP features were extracted. The following types of NLP features were extracted:

- word n-grams for n = 1 to 2
- character n-grams for n = 2 to 5
- syntactic dependency relationships
- review length ($log_2 x$ where x refers to the number of characters in the review)
- Brown corpus cluster IDs (for each word in the review text)

For the first feature type in the list above, otherwise known as "the bag-of-words" method, the review text is lower-cased and tokenized and then all one- and two-word sequences are identified. Likewise, for the character n-grams features, all two-, three-, four-, and five-character sequences are extracted from the raw, unprocessed review text. This latter feature type serves a couple different purposes:

- It extracts some textual features that would not be extracted via the n-gram features, such as emoticons, sequences of non-word (and word) characters, etc.
- It implicitly implements a basic form of automatic spelling correction as even misspelled tokens will still have large sub-strings in common with the correctly-spelled versions.

The third type of extracted feature, syntactic dependencies, extends the "bag-of-words" approach. The n-gram features represent sequences of characters/tokens and, for some simple cases (in English, at the very least), this suffices to extract a full syntactic structure. However, when syntactic structures are not sequential/linear or there is intervening content, such as embedded clauses, modifiers, etc., or the relationship simply manifests itself over a long stretch of words, the n-gram features are not enough. The syntactic dependency features are meant to cover such phenomena.

Lastly, the review length (as a base-2 logarithm of the number of characters, whose slope will decrease with review length) and a set of Brown corpus cluster IDs are extracted from each review. Although it is expected that review length may correlate weakly with experience (the more a user has used a product, the more he/she may have to say about it), the relationship is not necessarily linear. The difference between a review that is 10 characters long versus a review that is 100 characters in length is perhaps more significant than the difference between a 5,000-character review and a 10,000-character review. The Brown corpus cluster IDs are intended to capture a crude type of vocabulary distribution feature. Each token's cluster ID refers to a particular cluster of texts from the Brown corpus. Representing each review as a set of cluster IDs should provide some abstract idea about the content of the review.

All extracted NLP features (except for the length feature) are binarized, e.g., if a 2-gram such as "the game" appears five times in a review, the value of the extracted feature is converted to 1, signifying that that particular 2-gram appeared in the review. Features that do not occur in a review have an implicit value of 0. Thus, sparse vectors are used to create a vector space model.

The feature set used here is meant to approximate a sort of baseline NLP system. The main aim is to model reviews with a baseline system and then evaluate this kind of system's performance in terms proxies for user experience, including the amount of time the user spent using the product he/she reviewed (in hours), the number of times the review was marked as helpful, etc. If a basic system shows some potential in this regard, it may be taken as a sign that further feature set development could lead to better performance and, thus, that the main problem – filtering/sorting reviews by their truthfulness, helpfulness, etc. – can successfully be modelled.

5.2 User/review attributes

A set of user/review attributes is also extracted for use in the machine learning experiments. These features can be used in isolation, in combination with the NLP features, or not at all. For a non-exhaustive list of these attributes, see "Data" (Subsection 4.1).

Some of these features might typically be available for reviews and, therefore, could potentially be used to improve the performance of the modelled relationship between reviews and the phenomena related to reviewer experience. However, part of the aim in conducting experiments with these additional features is to explore the possibility that the relationship between reviews and reviewer experience could be successfully modelled by such features in isolation, i.e., without any NLP features at all. For instance, to what degree can knowing the number of friends a reviewer has tell how trustworthy his/her review is?

The experiments described below will explore each situation: models trained on NLP features, NLP features in combination with the review/user attributes, and the review/user attributes alone. Note that, when using review/user attributes to train a model to predict the number of times a review is marked as helpful, any review/user attributes that are related to the prediction label are automatically excluded to ensure that the model does not contain information that would definitely be related to what is actually being modelled. For example, if the prediction label is the number of times a review is marked as helpful, attributes such as the number of times a review was marked as unhelpful would be excluded.

6 Experiments

The aim of the experiments conducted as part of this exploratory work was to attempt to model reviewer experience and/or review usefulness in terms of the reviews and/or attributes of the reviewer/review.

6.1 `learn`

As part of `reviewer_experience_prediction`, the code repository that was developed for every phase of this work (which can be found at the following URL: `https://github.com/mulhod/reviewer_experience_prediction`), a utility was created called `learn`, which is used to conduct iterative machine learning experiments using `scikit-learn` under the hood. The motivation for making the experiments iterative was simply to try to reduce memory consumption: for each round of learning, only a small set of reviews, associated features, etc., must be stored in RAM at any given time. It is not unheard of for even a relatively small dataset of 1,000 reviews to require gigabytes worth of RAM after vectorization since the full feature set will reach into the hundreds of thousands. With the iterative learning algorithm used by `learn`, the user specifies the number of rounds/number of samples to use per round and the machine learning model is updated incrementally.

`learn` is the main entry-point of the system in terms of conducting machine learning experiments. Any set of games can be used for training (and possibly two different sets of games can be used for training/testing, a line of research that I hope to follow up on in the future). Seven machine learning algorithms from `scikit-learn` are available, including `Perceptron`, `PassiveAgressiveClassifier`, `PassiveAgressiveRegressor`, `SGDClassifier` `MiniBatchKMeans`, `MultiNomialNB`, and `BernoulliNB`. The aim in making this exact set of learners available was not necessarily to provide a representative sampling of popular algorithms, but rather it was due to constraints placed on the set of `scikit-learn` learners that could be used in an iterative fashion. Other aspects of the experiments that can be configured are the prediction label that is used (any of the reviewer/review attributes, e.g. number of game-hours played, number of times marked helpful, etc., see Subsection 4.1), the types of features used in the model (NLP features and/or a set of the reviewer/review attributes that are not directly related to the prediction label), and much more, such as whether to use "feature hashing" (further reduces memory consumption, but does not allow for subsequent model introspection, i.e., if the user wants to see the various model features and their weights), whether or not to generate the performance metrics for a "baseline" majority label system to provide a basis on which to compare the results, etc.

Reviewer/review attributes that are to be used as the prediction label must be manipulated before the learning process in order to allow for meaningful learning. For example, the values of the number of hours played attribute for any given game can range from 0 to upwards of 15,000 and, thus, it would not be a meaningful exercise to try to learn how the features map to this scale. `learn` makes use of a method of breaking down a scale such as 0 to 15,000 into various "bins", possibly of equal size, but not necessarily so. If the "bin" factor is 1.0, then the scale will be divided up into roughly equal-sized partitions and each of those partitions will be mapped to a new value, i.e., the position of the bin, with numbering starting at 1. For example, if 0 to 15,000 is divided up into three bins and the factor is 1.0, then it will result in the following type of mapping: [0, 5,000] → $1, [5,001, 10,000] \rightarrow 2, [10,001, 15,000] \rightarrow 3$.

Often, however, the raw values of the reviewer/review attributes are heavily positively skewed. For example, most reviewers have played the game being reviewed for less than 1,000 hours, but the distribution will have a long tail to account for the relatively small percentage of reviewers who have spent much more time playing the game. Thus, in some cases, it would be better to be able to include only 0 to 500 hours in the first bin, 501 to 1,500 in the next, and 1,501 to infinity for the last bin. The "bin" factor can be used to try to account for the highly positively skewed distribution of prediction label values. `learn` allows the user to specify how the label values should be "binned". At the end of the process, the predictions that the system makes will have to be understood in terms of this binning process. One can think of a new 1-3 scale as mapping roughly to a low, medium, and high experiential scale, for example.

More information on the distribution of label values (for hours played, specifically) can be found in the `GitHub.com` data-sets repository, a separate repository that is used to store only the data itself, which can be found at the following URL: `https://github.com/mulhod/steam_reviews`.

Game	Number of Samples		Number of CV Folds	
	Grid Search	Training	Grid Search	Training
Arma 3	500	2,500	5	10
Counter Strike	500	2,500	5	10
Counter Strike: Global Offensive	500	2,500	5	10
Dota 2	500	2,500	5	10
Football Manager 2015	240	500	3	5
Garry's Mod	500	2,500	5	10
Grand Theft Auto V	1,250	5,000	5	10
Sid Meier's Civilization 5	500	2,500	5	10
Team Fortress 2	500	2,500	5	10
The Elder Scrolls V	500	2,500	5	10
Warframe	500	2,500	5	10

Table 2: Number of samples/data folds used in each experiment (and, in particular, within each grid search cross-validation learning experiment and each main cross-validation training experiment).

Game	Accuracy	Precision (weighted)	F1 (weighted)
Arma 3	0.537	0.288	0.375
Counter Strike	0.621	0.386	0.476
Counter Strike: Global Offensive	0.707	0.500	0.585
Dota 2	0.655	0.429	0.519
Football Manager 2015	0.544	0.296	0.384
Garry's Mod	0.511	0.261	0.346
Grand Theft Auto V	0.689	0.475	0.562
Sid Meier's Civilization 5	0.781	0.610	0.685
Team Fortress 2	0.620	0.385	0.475
The Elder Scrolls V	0.598	0.357	0.447
Warframe	0.521	0.271	0.356

Table 3: Aggregated majority label system experimental results predicting the "game hours" label (a 3-point scale). Values represent the mean across all three experimental conditions. Because of slight differences from condition to condition in the data used, the order in which the data is used, and for what purpose the data is used, slight variations resulted across conditions, usually in the third or fourth decimal places at the most.

Game	Learner	Accuracy	Precision (weighted)	F1 (weighted)	QWK
Arma 3	MultinomialNB	0.501	0.426	0.444	0.080
Counter Strike	BernoulliNB	0.525	0.628	0.515	0.214
Counter Strike: Global Offensive	Perceptron	0.644	0.666	0.602	0.032
Dota 2	PassiveAggressive	0.589	0.509	0.544	0.169
Football Manager 2015	Perceptron	0.461	0.263	0.329	0.007
Garry's Mod	MultinomialNB	0.476	0.284	0.355	0.168
Grand Theft Auto V	BernoulliNB	0.415	0.617	0.418	0.108
Sid Meier's Civilization 5	Perceptron	0.680	0.663	0.670	0.007
Team Fortress 2	MultinomialNB	0.460	0.496	0.436	0.170
The Elder Scrolls V	MultinomialNB	0.479	0.526	0.481	0.066
Warframe	PassiveAggressive	0.519	0.496	0.493	0.091

Table 4: Average cross-validation performance metrics using the best-performing learning algorithm with all (NLP and non-NLP) features to predict the "game hours" label (a 3-point scale).

Game	**Learner**	**Accuracy**	**Precision (weighted)**	**F1 (weighted)**	**QWK**
Arma 3					
Counter Strike					
Counter Strike: Global Offensive					
Dota 2					
Football Manager 2015					
Garry’s Mod					
Grand Theft Auto V					
Sid Meier’s Civilization 5					
Team Fortress 2					
The Elder Scrolls V					
Warframe					

Table 5: Average cross-validation performance metrics using the best-performing learning algorithm with NLP features only to predict the “game hours” label (a 3-point scale).

Game	**Learner**	**Accuracy**	**Precision (weighted)**	**F1 (weighted)**	**QWK**
Arma 3					
Counter Strike					
Counter Strike: Global Offensive					
Dota 2					
Football Manager 2015					
Garry’s Mod					
Grand Theft Auto V					
Sid Meier’s Civilization 5					
Team Fortress 2					
The Elder Scrolls V					
Warframe					

Table 6: Average cross-validation performance metrics using the best-performing learning algorithm with non-NLP features only to predict the “game hours” label (a 3-point scale).

Game	**System**	**Learner**	**Accuracy**	**Precision**	**QWK**	**QWK-1**
Dota 2	majority label		0.93	0.86	n/a	n/a
Dota 2	NLP + non-NLP	MultinomialNB	0.96	0.96	0.65	0.65
Dota 2	NLP	Perceptron	0.96	0.97	0.77	0.78
Dota 2	non-NLP	Perceptron	0.93	0.86	0.0	0.0

Table 7: A selection of “number of times marked helpful” experimental results.

6.2 Experimental set-up

Machine learning experiments were conducted first with the number of game-hours played as the prediction label and then with the number of times a review was marked as helpful by users of Steam as the prediction label. Recall that the aim with these experiments was to model the amount of time a reviewer had spent playing a game using reviews and/or review/reviewer attributes. Each experiment involved ten rounds of iterative learning where a different set of 100 review samples were used in each successive round of learning. Additionally, a separate test set of 1,000 reviews was used to gauge the performance after each round of learning and at the end of the process.

Four of the available machine learning algorithms were used: `Perceptron`, `PassiveAggressiveClassifier`, `MultinomialNB`, and `BernoulliNB`. For each, a so-called "parameter grid" was used, which is a mapping between algorithm-specific hyperparameters and lists of hyperparameter values. In each learning round, every single combination of the hyperparameters was tried for each learning algorithm, the idea being to conduct the experiments with variously-tuned learners. Without diving into the subject too much, hyperparameters refers to tunings of a learner, such as the number of iterations, etc. The combination of machine learning algorithm and parameter grid that yielded the best performance metrics at the end of the process is reported in the results.

A number of different feature set combinations were experimented with: using the NLP feature set alone, the NLP feature set + the non-NLP feature set, i.e., the review/reviewer attributes, such as the number of friends that the reviewer has on Steam, and the non-NLP feature set alone. The aim here was to determine the role that the NLP features play in the modelling, whether or not the prediction label could be effectively modelled using only the non-NLP feature set, and whether or not both sets of features could be used in combination to produce a model that was more effective than that for either of the feature sets alone.

Lastly, the raw values for each prediction label were rescaled, i.e., the distribution of raw label values was forced into a scale from 1 to 3, with 1 signifying a low total number of game-hours played/number of times a review was marked helpful, 2 signifying a moderate total number, and 3 a high total number. Furthermore, the scale of raw label values was not merely divided into equal-sized partitions, but rather was divided up into increasingly larger scale partitions. For example, only 10% of the scale may be represented by the first "bin", 30% of the scale may be represented by the second and 60% by the last. This method of converting the raw label values to a three-point scale was used with both of the prediction labels in order to attempt to deal with the highly positively skewed label value distribution. For the great majority of labels in this data-set, the values tended to fall close to zero while the rest tapered off over a large range of values.

6.3 "Number of hours played" experiments

For the game-hours played experiments, three games were tested: Arma 3, Counter Strike, and Dota 2. Arma 3 and Counter Strike are generally alike in that they are military-based, first-person shooter-type games while Dota 2 is a so-called "MOBA" (multiplayer online battle arena) video game. In Table 2, a number of performance metrics are displayed for each game experiment's best learner (i.e., best combination of learning algorithm and hyperparameter set) and for a majority label-based system: accuracy, precision, quadratic weighted Cohen's kappa, and quadratic weighted "off-by-one" Cohen's kappa. Except for the majority label system, for which no machine learning was used, the type of machine learning algorithm is also presented. For Arma 3, results for each condition (only NLP features, only non-NLP features, and NLP + non-NLP features) are included while for Counter Strike and Dota 2 results are only presented for the NLP feature set-based experiments and the NLP + non-NLP feature set-based experiments.

In general, the non-NLP features fared surprisingly well on their own, but in general the NLP feature set-based experiments yielded better results in comparison. The combination of NLP features and non-NLP features often translated into lower accuracy in comparison to the NLP feature set-based experiments, but higher precision. In two out of the three cases, the majority label system resulted in the highest accuracy, but the precision of the majority label system was always improved upon by the feature set-based experiments, sometimes greatly so. In terms of the kappa metrics (for which no results can be given for the majority label system since, by definition, it always predicted only one label, which results in an undefined value), the Arma 3 experiments show that the feature set-based models could not predict the label at even chance level, but, for example, for the Dota 2 experiments, we see that the NLP feature set model actually can predict the label at a slightly-above-chance level (even if that level would be unacceptably low). For the Counter Strike experiments, observe that the NLP feature set-based model actually outperformed the NLP + non-NLP feature set-based model in terms of accuracy (0.82 to 0.6) and it came close to matching its performance in terms of precision (0.69 to 0.71).

Taken in whole, the results for the game-hours experiments do not reflect very well on the feature set-based experiments. They are often no better than the majority label system. While this fact may suggest that game-hours cannot be modelled with the feature sets, there are at least a couple indications of potential. On the one hand, some of the results show that the feature set-based models could at least to some degree perform at an above-chance level. On the other hand, the long-range perspective of these experiments is that they represent a real baseline system (the majority label baseline system that is used for comparison here is really but a poor choice for a baseline system as the decision is the same for each sample). Perhaps with better-designed features, performance will be able to be improved.

6.4 "Number of times marked helpful" experiments

For the experiments using the number of times a review was marked helpful as the prediction label, the results (presented in Table 3) are a little more favorable. Only Dota 2 was used for these experiments, mainly to get a taste for the kind of performance that might be expected for this prediction label. While the majority label system performs relatively well in terms of both accuracy (0.93) and precision (0.86), the feature set-based models actually perform even better (with the exception of the non-NLP feature set experiment, which matched the performance of the majority label system). Of particular note is the comparison between the NLP feature set-based model and the NLP + non-NLP feature set-based model: they both improve upon the majority label system in terms of accuracy (0.96 for both vs. 0.93 for the majority label system) and precision (0.97 and 0.96 for the NLP feature set-based system and the NLP + non-NLP feature set-based, respectively, vs. 0.86 for the majority label system). However, what is even more interesting is that the NLP feature set-based system outperforms the NLP + non-NLP feature set-based system in terms of the kappa metrics (0.77/0.78 for the NLP feature set-based system vs. 0.65/0.65 for the NLP + non-NLP feature set-based system). This additional boost in the kappa metrics signifies that the NLP feature set alone performs a great deal better than a system that uses the same features in addition to the review/reviewer attributes. In the case of the non-NLP features-only system, the system seems to have only learned the majority label as its performance largely matches that of the majority label system.

These results, while limited in scope and representativeness, signal at least some potential for effectively modelling reviews in terms of the number of times they were marked as helpful. Recall that this attribute of reviews is being interpreted as a proxy for reviewer experience and/or review trustworthiness. If the results hold for other games, then the further development of the feature set and more experimentation along this line of research could very well lead to a system that can accurately predict reviewer experience. Thus, it is conceivable that a system could be built that could factor into review filtering/sorting algorithms in order to improve the quality of reviews.

Acknowledgments

Again I would like to thank Janette Martinez and Emily Olshefski, graduate students at Montclair State University, for their help with the initial stages of this work. I would also like to acknowledge the advice and help I received from my advisor, Dr. Eileen Fitzpatrick, also of Montclair State University. Lastly, I would like to acknowledge the almost unlimited amount of computing resources provided to me by both Montclair State University and my employer, Educational Testing Service (ETS).

References

Bibliograpy

Nihar Jindal and Bing Liu. 2008. Opinion spam and analysis. *Proceeding of the International Conference on Web Search and Web Data Mining*, 219-230. ACM.

Tommaso Fornaciari and Massimo Poesio. 2014. Using Web-Intelligence for Excavating the Emerging Meaning of Target-Concepts. *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, 279-287. Gothenburg, Sweden. April. Association for Computational Linguistics.

Myle Ott, Yejin Choi, Claire Cardie, and Jeffrey T. Hancock. 2011. Finding deceptive opinion spam by any stretch of the imagination *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 309-319. Portland, Oregon. June. Association for Computational Linguistics.

Song Feng, Ritwik Banerjee, and Yejin Choi 2012a. Syntactic stylometry for deception detection. *Pr Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 171-175. Jeju Island, Korea. Association for Computational Lingusitics.