Bryan Adams
6.867 Machine Learning Final Project
December 4, 2001

Predicting Ebert Star Ratings

Discussion of the problem and approach

While most types of document classification tasks require many hand-labeled documents, movie reviews differ in two fundamental ways. First, reviewers typically "rate" the reviewed movie, creating a set of discrete classes. These ratings are typically between zero and four stars, with only one-half star of granularity, resulting in nine classes. Second, these classes, while discrete, can also be interpreted as a real-valued output. This orders the classes and creates "magnitude" for each of the classes. The work described below attempts to predict the star rating based on the text of the reviews.

This work is not terribly useful in its predictive power: since each review typically comes with a star rating, the prediction of this value will rarely provide missing information. For this reason, special attention will be paid to the process of the prediction, as novel information can be gleaned from which features most strongly predict the classification of the review.

Data collection and preprocessing

The following sections report on work done with a body of movie reviews by Roger Ebert (www.suntimes.com/ebert). Because the complete Ebert database of reviews is copyrighted, I created a small subset of reviews by randomly choosing reviews from five non-sequential years and manually copying them into a database (yielding 527 reviews). Ebert's star rating, as described above (`[0:.5:4]`) was also recorded for each review, as was the title.

In order to quantify the data, a word-detection scheme was applied to the data. Each detector returned the total number of appearances of a word in a review. It is worth noting that these counts are noisy, since the manual cutting-and-pasting routine concatenated words between paragraphs, obscuring some words and creating other new words.

The question of which words to search for is also relevant. For these algorithms, the set of detected words consisted of the union of all words in all reviews. This lead to an enormous dimensionality (8190 features) that includes many words one would assume to have no bearing on the star rating of the review. For this reason feature selection will be discussed in conjunction with each algorithm.

Kmeans

Justification

While clustering was not successful in predicting the ratings, it is instructive to examine this method's failure. I modified the traditional, unsupervised Kmeans algorithm to include supervision during the means update step. After the soft assignments were made, the new centroid was chosen to be the mean of all points that were both assigned to the centroid and belonged to the centroid's representative class (each centroid was initialized to be from one of the classes – see the discussion below). This was to encourage clusters around the less popular ratings (since the vast majority of the movies were rated 2.5 or 3 stars).

Feature selection

The feature selection problem is extremely difficult when using a clustering algorithm. Because the algorithm is iterative and highly dependant on the initialization, it is not typically feasible to run it many times as a part of a wrapper approach. However, a filter approach that will select the right features is not readily apparent. The features leading to optimal clustering are those that will cause samples from a particular class to be close together (in Euclidean distance) but far from the rest of the samples. This double requirement is not easy to satisfy.

For lack of a better method, I chose to filter the data using the mutual information between a class and the data. The feature selection process should look separately for features are highly correlated with each particular cluster. Taking this cue, the mutual information was computed using the following data:

- The target vector is set to one for a particular class
- The data is set to 1 if the number of words in a given review is greater than or equal to the average number of time that word appeared in all reviews.

Results

As stated previously, the results for this supervised Kmeans method were disappointing. The testing was done by running the algorithm on all 527 points and then correlating the data with the given labels. Reports here will be on a simplified version of the task: rounding each star rating down (creating 5 values: 0, 1, 2, 3, or 4 stars).

The initial test was to simply cluster the data around two centroids, ideally separating the "good movie" reviews from the "bad movie" reviews. The features were selected using the mutual information method discussed above, resulting in 40 features: 20 features with the highest mutual information with 0 or 1 star reviews and 20 features with the highest mutual information with 4 star reviews. This yielded some success (correlation matrix follows):

```
a =
    10    29    43    35     1
    10    46   102   213    38
```

In this case, the 4 star reviews were almost all in the second cluster and the 2-star and 3-star reviews were reasonably correlated with the second cluster. The zero- and one-star reviews, though, were too evenly split to predict. This is particularly interesting, given that they were a part of the feature selection process.

As the number of clusters was boosted to the full 5 (ideally, one for each star-rating), the performance dropped:

```
a =
     1     3    28    91    20
     4    23    39    68     5
     5    23    37    53     4
    10    26    39    33     1
     0     0     2     3     9
```

Discussion

The random nature of the algorithm leads to a heavy dependence on the initialization of the centroids. I tried several different initialization methods (see code for details); the method where each mean is a randomly chosen point from each class tended to produce the best results (and it was used in all tests). Note that it relies on some supervision, which probably gave it an advantage over the other methods. This method, though, implies no more than one centroid per class. While some preliminary results showed that larger numbers of clusters (with unsupervised initialization) did not improve the results, it is possible that some combination of supervised initialization and large numbers of clusters would be more successful.

It is also possible that the clusters existed, but my supervised algorithm for updating the means was unable to find them. Adding supervision to the algorithm almost certainly would have helped regularize the data for better performance on a potential test set as well. However, the resulting clusters in both the supervised and unsupervised environments were too mixed to allow for any meaningful prediction.

Another method that was promising but too computationally expensive was a clustering technique using a genetic algorithm to search the space of the means. This was implemented, but did not generate notable results in the few number of generations I was able to run. However, this method is well-suited to this particular problem, because the supervised aspect of the fitness function allows the means to be tilted in favor of purity of cluster, which may have led to a higher correlation scoring.

I believe the clustering method performed poorly for two main reasons. First, misclassification is not penalized by the magnitude of the mistake. Since the clusters do not have a "value" that corresponds to the star rating, placing a review in a 4-star cluster instead of a 0-star cluster is simply counted as one mistake. While this could perhaps be corrected algorithmically, at the most basic level, there is little hope for this method. Reviews with poor ratings do not necessarily contain similar words, and so there is little reason to hope that they will be close together in this data representation. This makes clustering was a bad choice for this data set.

Naïve Bayes

Justification

The naïve bayes method is very commonly associated with document classification tasks. Because naïve bayes deals with binary values, the data was initially simplified into two classes: Ebert's well known "thumbs-up" and "thumbs-down" rating system (corresponds to a star rating over and under two-and-a-half). He notes to his readers

that this is the cutoff for a movie to be "recommended", which would seem to indicate a significant difference in word choice between these two classes.

Feature selection

The feature vectors were reduced to binary values in two ways for comparison. The first is a simple word detector – the feature is set to "1" if the word is present in the review. A mutual information test was run over each feature with the binary target vector (thumbs up/down), and naïve bayes was run over different sizes of feature vectors to determine the optimal length (it turned out that the minimum mutual information to improve the results was approximately 0.005, creating a feature space with 1297 features). The list of the ten highest-correlated word vectors reveals what the method was looking for:

'DRAMA'
'THEIR'
'HEARTWARMING'
'FEELINGS'
'ATTRACTED'
'RECKLESS'
'DEGREE'
'DIMENSIONAL'
'AWARD'
'RECYCLE'

These are all fairly common words (except for "heartwarming," "dimensional," and "reckless," all were in the top 1700 in frequency out of 8190). Many of them are typically associated with extreme movies: "heartwarming," "award," and "recycle" would be expected in movies that arouse strong feelings (negative in the case of "recycle").

Results

This method, when carefully optimized, could successfully predict the "thumb-rating" of a movie with up to 87% accuracy. It is interesting to note that the "presence" feature detectors worked considerably better than the "more than average" feature detectors, and the results reported here will all be from that data set.

In the initial testing round, the naïve bayes parameters were trained on the first 400 data points and tested on the remaining 127 data points. The resulting log-likelihood and mean prediction error over the test set is as follows:

```
[llike,mpe] =
 -114.7139    0.1339
```

A closer look at the errors reveals how the data can be very difficult to predict from. The titles and star rating of the mis-predicted reviews are as follows:

Stars	Title
2.5	'tough guys don't dance '
2.5	'les miserables'
2	'no looking back'
3	'barney's great adventure'
3	'grease'
2	'twilight'
1	'taste of cherry'
2	'palmetto'
3	'the borrowers'
2	'four days in september'
3	'star kid'
2.5	'the winter guest'
2	'swept from the sea'

In this table, there is only one misprediction that has a magnitude over 1 star: "Taste of Cherry," a widely-hailed movie from Iran which Ebert rated poorly. In his review of the movie, Ebert spent considerable space discussing how other critics saw the movie. Words such as "masterpiece," "ovation," and "fascinating" were mixed in with "boring," "excruciating." This demonstrates how the lack of semantic knowledge can make this task impossible.

Discussion

Naïve bayes was extremely sensitive to the regularization parameters. The parameters that result in reasonable prediction must be carefully tuned. Small deviations (a little as 1/100 in the sample size probability) could lead to drastically reduced results.

The other concern with this method is the limitation in expression. Because the method only uses binary information, the potentially useful extreme values (words associated with a zero-star review versus a 2.5 star review) are lost. One could extend the method to categorize into multiple classes, however an additional weighting scheme that penalizes "big misclassification" (e.g. 0-star to 4-star) would need to be added.

Linear regression

Justification

Linear regression, while probably the simplest method, led to the best results. This method is particularly well suited to the star-prediction task, because the predicted outputs will be real-valued, and therefore will take advantage of the real-valued quality of star ratings. It is worth noting that this is still a classification task (the prediction from the regression is classified according to its nearest half-star value), though the method can easily be adapted to changes in the granularity or scale of the star-rating process (e.g. a quarter-star rating or 5-star ratings).

Feature selection

Because the optimization of the weights is a (relatively) fast process, feature selection is geared towards improving performance instead of reducing computation. In this case, a greedy feature selection method was used to pick from among the many words: features were added one-by-one according to which would create the lowest new training error.

The result of this greedy feature addition was curiously different from the mutual information criterion (again, the top 15 words only are listed):

S
FUNNY
PERFECT
GIRLS
KNOWS
GO
TOWN
GIVES
HUMAN
D
THINGS
EVERYTHING
MOMENTS
V
SAYS

Some of these words have high qualitative meaning and are expected: "funny," "perfect," "moments." Also in the top 30 are words such as "problem," "important," and "manfully" (a word Ebert frequently uses as a jibe at action features). The word fragments point out a general bug in the system: a word like "go" will be found in all instances of "gone" as well as "going" or "gordon." Looking further down the list, names like "martin" and "pesci" appear in the top 35, since Steve Martin and Joe Pesci are included in the reviews, have names that will only appear in those reviews, and are typically associated (in this body of reviews, anyway) with very good or very bad movies.

Results

The regression was able to accurately predict the star ratings for the movies with reasonable accuracy. Due to computational constraints, the final training vector had only 200 features, and it was chosen from only the first 3000 possible features. Under these constraints, the mean star misprediction in the training and test sets is as follows:

```
[train_e,test_e] =

    0.2706    0.3150
```

The following is a list of the movie titles that were mispredicted by one full star (the maximal value):

3.5	'the fourth protocol '
1.5	'the secret of my success '
2.5	'the untouchables '
3.5	'the witches of eastwick '
3.5	'wish you were here '
3.5	'character'
3	'the leading man'
3	'the borrowers'

Of these eight movies, it is notable that three, "the fourth protocol," "wish you were here," and "character" are well-reviewed but serious dramas about "negative" subjects (nuclear war, child abuse, father-son hatred), which probably skewed the language towards more negative terminology.

Discussion

The domain for the linear regression problem was the most complicated, yet the method still performed well (and would have outperformed the naïve bayes method predicting thumbs up/down). Because linear regression does not count on each "class" having similar characteristics, additional features were able to improve performance. Regression was also able to best utilize features that affected very few reviews, but affected them strongly (such as the actors' names) because an additional sparse dimension would not have an effect the reviews for which its value was zero. With more computational power, more features would have performed even better, since the number of learned parameters is still fairly low.

This method was successful, ultimately, because it was the best match with the data. Since each review is represented by the total appearances of each word, all semantic data is lost. It is not unreasonable for two reviews with very different ratings to have similar word counts, so the data is extremely densely packed and, in many ways, inseparable. Linear regression does not weigh any one data point too heavily (as support vector machines would be likely to do), and therefore allows for good classification.

Conclusions

While the performance of naïve bayes and linear regression were admirable, several reviews proved to be difficult to classify under any circumstances. Quantifying the data, a challenge in any learning project, was particularly difficult here. Another method that would be more sensitive to the patterns of words or semantic meaning of the words (such as hidden markov models) would have a significant advantage over any method that simply counts appearances of words.

I believe that these methods were also hindered by the noise in the data. Many of the "words" the features were detecting were single letters that errantly appeared in the reviews, but were then counted many times. Substring matches are another example of noisy data: if the word "hate" is an indicator for poor reviews (as one would expect it to be), the word "hat" would have a very similar feature vector, perhaps winning out in a feature-selection process and obscuring the real relationship between "hate" and poor reviews.