CS229 Project Report: Recommending movies and TV shows based on Facebook profile data

Benjamin Paterson (paterben@stanford.edu)
Weifeng Zhang (wfzhang@stanford.edu)
Tim Mwangi (tmwangi@stanford.edu)

December 13, 2012

Abstract

The aim of this project is to learn how to make predictions on what genre of movie a user is likely to be interested in based on their raw Facebook data. We collaborated with other groups to collect a dataset of 10k unique users which was then parsed, stemmed and tokenized. We trained two predictors each using a different model, SVM and Naive Bayes. In order to correct for the skew in the training sets, we generated an independent unskewed dataset that was a subset of the original data, and compared performance of the classifiers using several metrics (accuracy, MCC, AUC). We then attempted several improvements, first to the Naive Bayes model itself by using TF/IDF weighting and document vector length normalization, second to the features themselves by filtering tokens using Mutual Information. The combination of TF weighting and feature selection yields significantly better performance in terms of MCC and AUC.

Introduction

The Facebook graph API was used to scrape friend Facebook profile data. This results in a dataset of Facebook user profiles in XML format, listing for each user the gender, locale, 'about', liked athletes, teams, books, TV shows, movies, music, activities, interests and sports sections of their Facebook profile. The 'liked' TV show and movies sections act as the labels for our training and test data and the rest of the sections are used as the attributes.

With several groups working on this project around 10k users worth of Facebook profile data was collected. A table associates 347 movies (title and description drawn from IMDB and Wikipedia) with their associated movie genres (88 unique genres). User Facebook movie and TV show 'likes' are then compared with the table to generate labels for each user profile. These labels are represented in $y \in \{0,1\}^{88}$ as each user can like or not like each genre.

Raw Data Processing

The fields of raw data (interests, about, etc.) for each user are concatenated into a single string and then stemmed using the Python NLTK library. Punctuation, stopwords, non-alphanumeric characters are eliminated to produce a single list of about 32000 numbered tokens which acts as the vocabulary. Each word in each profile is converted to the index of the corresponding token and the resulting matrix is stored in a file. The labels for each user are computed by comparing the table mapping movie titles to genres with the movie 'likes' of each user through simple string matching, and are then stored in a separate file. Users with empty profiles or with 0 'liked' categories are eliminated, shrinking the dataset from 10k to about 3.6k users.

As a result, the features are a vector of about 32000 tokens, $x \in \mathbb{R}^{32000}$ and $x_i \in \mathbb{R}$ where x_i is the frequency of that token in the user profile. Each line in the feature file corresponds to a user profile and lists the indexes of the profile's tokens in the vocabulary. Each line in the labels file corresponds to the categories assigned to the user based on the movies and TV show information present in the user's profile. The most popular categories in order are: comedy (8671 likes), goofy (6645 likes), satire (4454 likes), drama (4352 likes) and debauchery(3419 likes).

We decided to take a multi-label classification approach to this problem, that is we treat placing a user into a category as a binary problem, and treat all categories independently. For each category c, $y \in \{0,1\}$ and $y = 0$ if the user does like any movie in that category and $y = 1$ if the user likes at least one movie in that category.

Methods Attempted

We trained a Naive Bayes classifier and an SVM.

Naive Bayes

Basic Naive Bayes Classifier

Our first approach was to train a Naive Bayes classifier using the multinomial event model with Laplace smoothing. Since we are doing multi-label classification, we trained one binomial classifier per category, ie. classifier i tries to predict whether a queried user profile will have liked category i. In order to test the classifiers we split the user profiles randomly into training set (70%) and testing set (30%). Some results for these classifiers are shown in fig. 3-5. Precision, recall and error are all globally increasing functions of the number of users interested in the classifier's category (number of labels $y = 1$) as long as the average number of users interested is less than 50%. After this point, error goes down and precision and recall continue to increase. This suggests that the classifier is strongly affected by the prior distributions of the labels. We come back to this later. In order to get a performance metric that is not biased towards one type of label (high precision and recall are not good indicators of performance when the data has a lot of positive examples), we use the MCC:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

MCC values are between -1 and 1, with 1 indicating perfect classification and -1 complete misclassification. Naive Bayes achieves a low average MCC of 0.029 (see Fig. 3).

The MCC values seem to have the same monotonicity as the error rate when plotted against the priors, with a maximum around $p(y = 1) = 0.50$. The values are overall very low. In order to know whether the problem came from the feature space, we tried using a smaller vocabulary of 5000 tokens using the movie description data. This new feature space did not improve performance.

Unskewed Naive Bayes

Since the prior distribution of the labels seemed to have a strong effect on the classifier, we trained a second Naive Bayes classifier using unskewed training data. This data was selected randomly and independently for each category so that the training set had an exact 50/50 distribution of positive and negative examples. We compared the two versions of Naive Bayes using several metrics, averaged across all 88 categories: MCC, the area under the ROC curve (AUC), error rate, precision and recall. The results are detailed in Table 1.

On average, Unskewed Naive Bayes makes more positive predictions as the bias in the training set, which is negative for most categories, is no longer present. As expected, recall is much increased at the cost of the error rate. Perhaps more importantly, MCC and AUC are both higher in the unskewed version.

Naive Bayes improvements

[2] suggests using a "transformed" version of Naive Bayes using the three following operations:

1. Transform the raw word frequency d_{ij} of word i in document j by $d_{ij} := log(1 + d_{ij})$ (TF transformation)

2. Normalize word frequencies by their inverse document frequency $IDF_i = \frac{\sum_j 1}{\sum_j \delta_{ij}}$ where $\delta_{ij} = 1$ if word i appears in document j (IDF transformation)

3. Normalize word frequency vectors to length 1 for each document: $d_{ij} := \frac{d_{ij}}{\sqrt{\sum_k (d_k j)^2}}$ (L normalization)

The paper also suggests improvements to alleviate the effects of skewed training data, in particular using the complement class, but they have no effect in the case of binomial classification.

We implemented all possible combinations of the above three transformations, on both basic and unskewed Naive Bayes. On the basic version, we found that length normalization when paired with the TF and IDF transformations often gives uniformly negative predictions, but each transformation works acceptably on its own. None of the transformations yielded a significant improvement on the unskewed dataset Naive Bayes. However, applying the TF and IDF transformations on the basic dataset yielded significantly better results in terms of MCC, AUC and recall at the cost of accuracy.

Feature Selection

Description

We attempted feature selection using an adaptation of the Mutual Information technique discussed in class. Given the 88 categories and about 32000 tokens (features), for each category we computed the mutual information of that category per token using the formula supplied in the class lecture notes:

$$MI(x_i, y) = \sum_{x_i \in \{0,1\}} \sum_{y \in \{0,1\}} p(x_i, y)\ log\ \frac{p(x_i, y)}{p(x_i)p(y)}$$

The probabilities $p(x_i, y)$, $p(x_i)$ and $p(y)$ were computed from the empirical distributions and we used Laplace smoothing to avoid dividing by 0 where applicable. We got an 88 by 32000 matrix and then we computed the columnwise average to obtain the average mutual information and used it to select features that have above a certain mutual information score since the features with a score equal to or higher than this score are the most highly correlated with the categories. We used a threshold MI score of 10^{-5} to reduce the number of tokens from 32000 to just 1019. The features removed are those tokens that are very infrequent throughout the training set. For example tokens that appear only once in a single training example will have very low correlation with the categories. On the other hand, tokens that are very frequent like NUMBER have a very high correlation with the categories and hence have high Mutual Information. Thus it makes sense to discard those tokens that do not have a high correlation with the categories.

Application to Naive Bayes

Applying Naive Bayes with this new document space was simply a matter of applying the MI filter after the tokenization step and running the resulting dataset through the Naive Bayes pipeline. Again, two datasets were generated, one skewed and one unskewed, and all the combinations of TF weighting, IDF weighting and length normalization were tested on the two datasets.

Below is a plot that shows the MCC difference on a per-category basis between the basic versions of Naive Bayes (no transformation, skewed data) on the raw dataset and on the MI-filtered dataset. The plot shows that in most of the movie categories, there was an improvement in the MCC after performing feature selection. Furthermore, since there are less features involved in training and testing Naive Bayes, running the algorithm takes much less time and space.

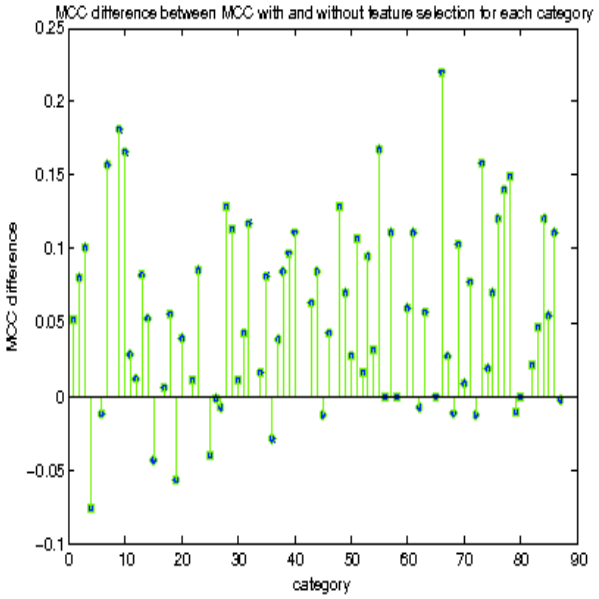

Fig. 1: MCC comparison for basic Naive Bayes with no filter feature selection applied and with filter feature selection applied.

With feature selection, the performance of basic Naive Bayes in terms of average MCC jumped from 0.029 to 0.093, and in terms of average AUC from 0.442 to 0.545, with only a small loss in terms of accuracy. However, the best results were obtained using TF weighting, with the TF/IDF combination close behind. These results are shown in Table 1. Feature selection did not seem to have a positive effect on the unskewed version of Naive Bayes.

In terms of AUC, the unskewed, non MI-filtered classifiers remain the best performers. However, we believe MCC is a better metric than AUC for the same reasons we preferred MCC over precision and recall.

SVM

We trained an SVM in parallel with Naive Bayes.

Scaling

We used scaled data to avoid tokens with greater numeric ranges dominating those with smaller numeric ranges, since kernel values depend on the inner products of feature vectors. It is also helpful to simplify calculation by scaling. We scaled data to [0,1] for both test and training data.

Kernel selection

Commonly used kernels are the linear kernel, $K(x_i, x_j) = x_i^T x_j$ and the radial basis function (RBF)

Metric	Basic	TF/IDF	L	Unskewed	MI	MI + TF/IDF	MI + TF	SVM linear	SVM RBF
MCC	0.029	0.050	0.052	0.051	0.093	0.100	0.112	0.009	0.083
AUC	0.442	0.513	0.400	0.676	0.545	0.561	0.564	-	-
Error rate	0.123	0.238	0.101	0.618	0.148	0.177	0.126	0.166	0.100
Precision	0.161	0.164	0.597	0.107	0.206	0.196	0.242	0.146	0.450
Recall	0.141	0.336	0.081	0.879	0.260	0.329	0.224	0.167	0.087

Table 1: Comparison of different versions of Naive Bayes (first 7 columns) and SVM (last 2 columns).

$K(x_i, x_j) = exp(-\gamma||x_i - x_j||^2), \gamma > 0$. We used both to find the boundary for each category.

Cross Validation for parameter

To find the penalty parameter(C) and gamma for the model, we used the Grid-search method with 5-fold cross-validation. Grid-search is time-consuming. However, it is good for parallel computing, since each pair C and gamma are independent.

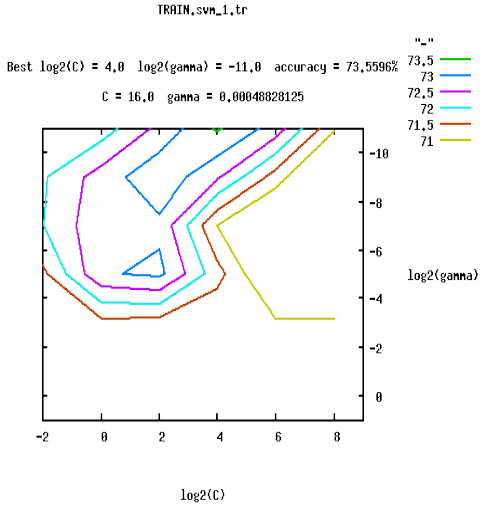

Fig. 2: Grid search for parameter in SVM with 5-fold cross-validation.

The result above shows that for category 1, C=512 and $\gamma = 3.05 \cdot 10^{-5}$ is preferred for the RBF model. For the linear kernel, we used the L2-regularized L2-loss linear support vector classification model .

Results

RBF has better performance than Linear kernel except in recall rate from Table 1. RBF nonlinearly maps the features to an infinite dimensional space, so it has more flexibility than the linear kernel hyperplane. Furthermore, linear kernel is a special case of RBF [3]. We calculated the ratio of likes in test set and likes in training set and plotted the MCC against it (see Fig. 4). The best score happens at ratio 0.42, which is similar to $\frac{test\ set\ size}{training\ set\ size} = \frac{30}{70} = 0.4285..$ This indicates that SVM's performance is better when training data is less skewed, similar to Naive Bayes.

Comparison between Naive Bayes and SVM

For both Naive Bayes and SVM, MCC is large in categories with many likes, while error is small in strongly skewed categories. RBFSVM's MCC is comparable to the best performance by Naïve Bayes, though slightly lower.

The SVM approach seems to yield the best accuracy (90.00%). Some of the Naive Bayes approaches (length normalization only, for example) yielded similar accuracy but at the cost of increased MCC. Naive Bayes with MI filtering and TF transformation still performed best in terms of MCC while maintaining an acceptable error rate and AUC.

Conclusion

The final performance of our classifiers is much better than the basic Naive Bayes and linear kernel SVM that we first implemented. However, an MCC score of 0.112 is still very low compared to the best theoretical performance of 1. The sparsity of the tokens in the user profiles and the uniformly negative predictions of our classifiers on some categories suggest that more data per user profile may yield significant improvements, for example by incorporating status updates in the user profile data. However, we believe that the biggest culprit is the way the data is labeled: there is no guarantee that a user that does enjoy a particular category of movie has actually liked a movie from that category on Facebook.

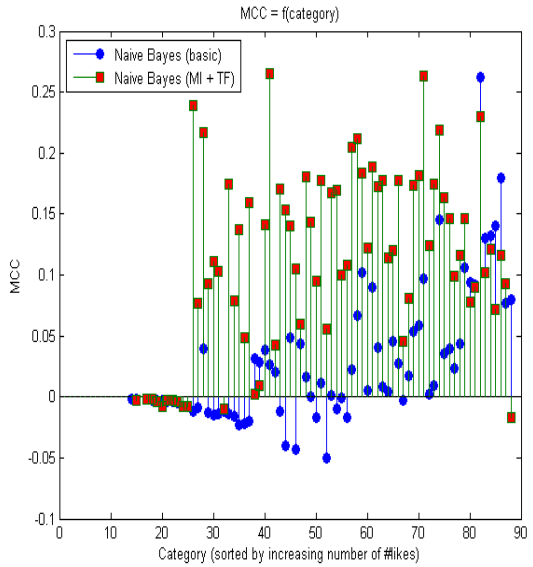

Fig. 3: A plot of the MCC of basic Naive Bayes and Naive Bayes with MI + TF for each category. The latter has consistently higher MCC.

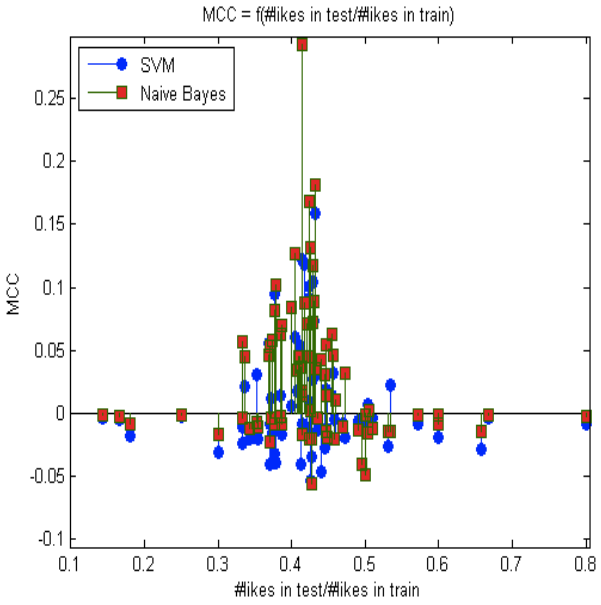

Fig. 4: A plot of MCC against the number of likes in the test set divided by the number of likes in the training set. Both SVM and Naive Bayes do best when the training and test set have similar label distributions. This was one of the motivations for creating an unskewed dataset.

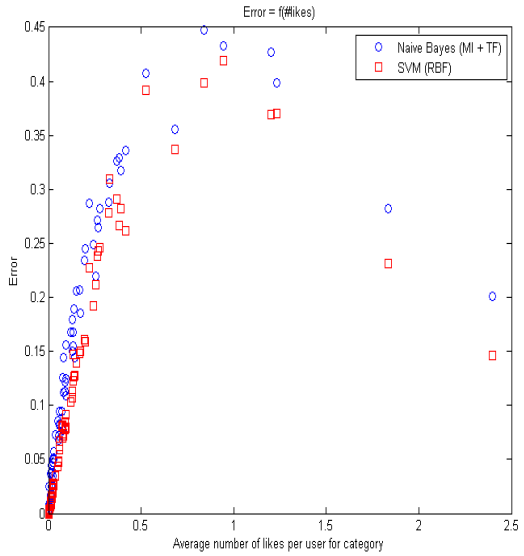

Fig. 5: Error rate as a function of the average number of likes per user for a category for both Naive Bayes and SVM.

Acknowledgements

We would like to thank the ScreamingVelocity founders, Wayne Yurtin and Graham Darcey who came up with this project and provided the initial data scraping scripts, as well as Andrew Maas and the rest of the CS229 course staff for their guidance while we were working on the project. We are also grateful to every one of the students in CS229 who provided their own Facebook data.

References

[1] Fan Rong-En, Chang Kai-Wei, Hsieh Cho-Jui, Wang Xiang-Rui, Lin Chih-Jen. LIBLINEAR: A Library for Large Linear Classification. Journal of Machine Learning Research 9 (2008) 1871-1874. Submitted 5/08, Published 8/08.

[2] Jason D. M. Rennie, Lawrence Shih, Jaime Teevan, David R. Karger. Tackling the Poor Assumptions of Naive Bayes Text Classifiers, In Proceedings of the Twentieth International Conference on Machine Learning, 2003, pages 616–623.

[3] Keerthi, S. S., C.-J. Lin. Asymptotic behaviors of support vector machines with Gaussian kernel. Neural Comput. 15 (7), 2003, pages 1667–1689.