What's So Funny?

IS622 Final Project
Brian Chu | December 20, 2015

Overview

Sense of humor is a tricky, if not impossible concept to quantify. Some argue that it is an abstract concept that cannot be measured, much like love and happiness (1). Others propose that context clues (2), cultural norms (3), or even biological traits (4) contribute to the formula of funny.

In this project, I look at sense of humor from the perspective of written jokes and try to see if commonalities can be discerned using collaborative filtering and content-based recommendation system techniques. A secondary objective is determining whether sense of humor can be better predicted by similarity with other persons or by specific joke features. Finally, I aim to improve on common recommendation system models by investigating a multi-stage approach using clustering.

Methodology

Dataset and Preparation

I used the Jester dataset (5), which includes 1,805,072 ratings from up to 100 jokes by 24,983 users. On average, each user rated 72/100 (72%) of jokes and each joke was rated by 18,172/24,983 (72.7%) of users. All users and jokes had at least one rating, with the range being [36, 100] and [8532, 24,981], respectively.

The ratings range from -10 to 10, but were normalized to 0 mean and unit standard deviation for unbiased comparisons of users. Jokes that were not rated by a user were reclassified from 99 to 0 so not to influence any distance or squared error calculations.

The full corpus of jokes can be viewed here. These were initially 100 separate html files, which were scraped using the BeautifulSoup package. For later text classification, a copy of the corpus was used that was stripped of common stop words, punctuation, and case sensitivity using the NLTK and String packages.

Tools

PySpark v1.5.1 was predominantly used for machine learning tasks (MLLib package) and linear algebra calculations while Python v2.7.9 was required for some data wrangling and summarization (most notably scikit-learn and pandas packages). All coding was done through the Jupyter Notebook and can be viewed here or the entire Github repository here. Please refer to those sources in lieu of some brevity presented in this report.

Analysis

The initial analysis is based on two fundamental recommendation system techniques: collaborative filtering and content-based, often referred to as CF and CB in this report and described first here. A secondary analysis involving clustering will be described in a later section.

Collaborative filtering (CF)

Collaborative filtering is a method that focuses on the similarity of users and ratings for common items to make recommendations. The matrix representation for CF has users on the rows and item (joke) ratings on the columns, which is exactly how the Jester dataset is setup. Therefore, no additional data wrangling was required for CF analysis other than the normalization process described earlier.

CF analysis was implemented with PySpark's MLlib package, which uses the Alternating Least Squares (ALS) algorithm to predict missing entries. ALS aims to minimize the squared error of the UV user-item rating matrix by alternating learning sequences of U to V and V to U until a steady state has been reached. This is similar to the collaborative filtering theory discussed in Chapter 12 of *Mining of Massive Datasets* (6).

To train the model, MLLib ALS requires a Ratings object, which is a tuple in the format (userID, itemID, itemRating). The Jester dataset was, therefore, partitioned as a PySpark distributed dataframe, parsed for only rated items, and converted to the proper format using the rdd.flatMap function. This ratings set was then divided into a 75% train and 25% test set to evaluate its initial performance. The CF model was then applied to users' non-rated jokes to recommend the top-rated ones based on predicted rating.

Note: Several parts of this CF section were built on my Week 12 mini-project

Content-based (CB)

In contrast to CF, content-based methods focus on the similarity of item features rather than users to make recommendations. CB requires both a user profile matrix of his/her preference for specific features, as well as an item profile matrix that represents the degree of inclusion of those features for each item.

For the Jester dataset, item features were words in the text of the joke. As mentioned earlier, the text used here was already stripped of stop words and punctuation. To determine the feature degree of each joke, the term-frequency-inverse document frequency (TF-IDF) method was used. In brief, TF-IDF is a weighted measure that calculates the relative importance of a word to a document while counterbalancing the frequency of that word to the corpus of all documents. PySpark's built-in TF-IDF function was used to extract item features from the corpus of jokes, which resulted in a distributed set of 100 sparse vector representations. These vectors represented the item profiles for CB recommendations. To create the user profile, the same TF-IDF calculation was done for each of his/her rated jokes

but weighted by the specific rating for that joke. The aggregation of these vectors then formed the overall matrix representation of his/her feature preference.

Finally, to put the user and item profile together and gauge similarity, the cosine similarity between both matrices was calculated for each user-item pair. The top recommendations were the set of unrated jokes with greatest cosine similarity between user preferences and item features.

Initial Results

CF accuracy

CF model accuracy was measured using the trained ALS model on the 25% test set. The results are encouraging with an MSE and RMSE of 0.009 and 0.079, respectively. Perhaps more informative is the mean absolute error, which is 0.076. Since the error measures are already based on normalized data, the MAE does not need to be further adjusted as in the Goldberg paper (5). In that paper, normalized MAEs of 0.187-0.237 were reported, but a more fair comparison may be made using my Week 12 mini-project, which did not pre-normalize the data and reported an MAE of 0.165. In either scenario, it appears the ALS model trained here is very good.

CB accuracy

Training and test sets are less conducive to CB systems where the outcome was item similarity based on item features rather than predicting actual user ratings. Therefore, we can look at the highest cosine similarity items compared to the actual rated and recommended ones. In fact, that is what's done in the next section to compare CF and CB.

CF vs. CB accuracy

Comparing CF and CB accuracy requires a separate approach because of the different measures used for each independently. Other studies have previously used precision, recall, and F1 scores to compare recommendation systems (7,8), which is the method I incorporated here. For each user, I split out 25% of their rated jokes as the test set and predicted the top ratings using CF and top similarity scores using CB. These recommendation sets were then compared to the actual ratings of the test set to determine precision, recall, and F1 score.

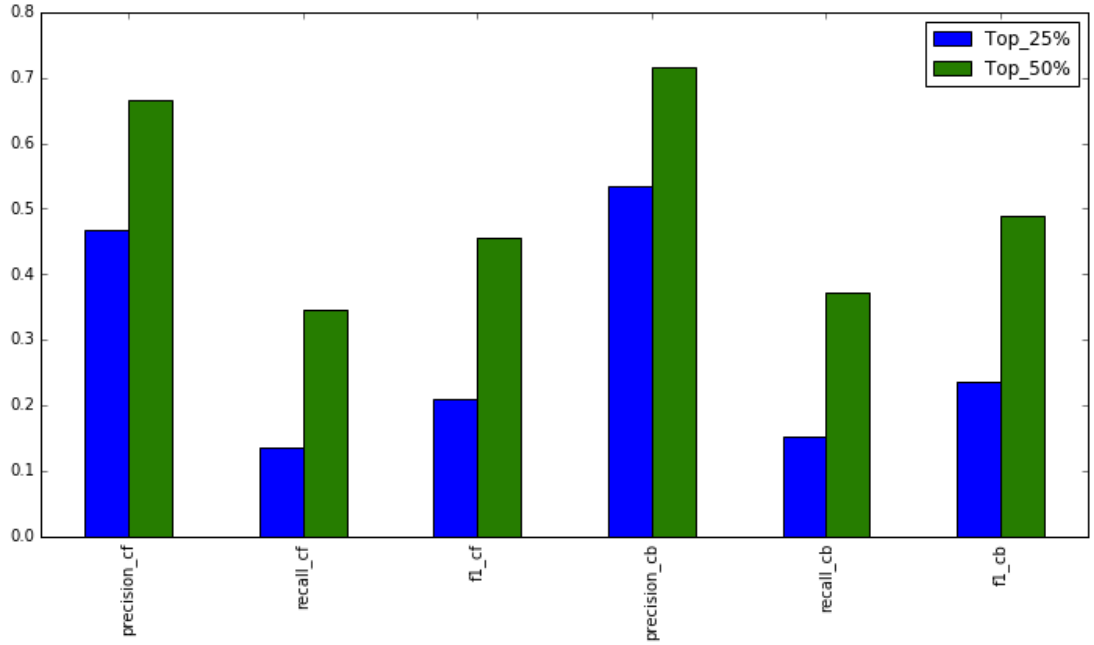
Precision *= Number of correct matches / Number of possible matches*

Recall *= Number of correct matches / Total number in test set*

F1 score *= (2 x precision x recall) / (precision + recall)*

These accuracy measures were calculated for both *including* ranked order of the recommendation set and *excluding* ranked order. For practical purposes, I believe the unranked scores are more indicative as it eliminates small ratings variance or standard error. Therefore, the results below are for *unranked* accuracy only. I also assigned even weights to each user rather than weight them by the number of unrated jokes.

For predicting the top 25% and 50% of recommended jokes, the results were:



Precision for CF was about 47% and 67% depending on N, while for CB it was 54% and 72%. Recall, as it is calculated, is a tougher estimate to assess since the notion of false positives here is different than in most use cases. F1 score, as the harmonic mean of precision and recall may be a more appropriate accuracy statistic if factoring in recall. It also makes sense that precision would increase and recall decrease as N% increases. Therefore, again, F1 may be a favorable measure if N is to be disregarded.

The results were admittedly surprising. I was not expecting CB to be as accurate as CF, let alone *more* accurate on all measures. When initially eyeballing the joke text, common words were not so obvious but apparently TF-IDF did a very good job at extracting the important features. These results might also suggest that while a user may find several types of jokes funny (or unfunny), there is more consistency in ratings within users than between users.

Clustering

The surprisingly strong results led me to wonder if a multi-stage approach that initially filters down users or jokes would improve the recommendation system model. In particular, I was interested if clustering methods learned in this class could identify 'neighborhoods' that either a CB or CF recommendation system would then solely choose from. This follows the thinking in MMS 9.3 as well as some literature exploring similar concepts (9,10).

A two-stage clustering and recommendation system model could work as:

1) Cluster *items* + CB
2) Cluster *users* + CB

3) Cluster *items* + CF
4) Cluster *users* + CF

Logically, it did not make sense to combine a clustering method with a CB recommendation system. This is because CB isolates a user-item feature matrix so regardless if you pare down the number of users or items, the feature set for any one item and user remains the same and the model does not change. In addition, the algorithm still has to pass through all the jokes so runtime remains at *O(N)*. Therefore, both options 1 and 2 were eliminated.

Option 3 of clustering items and then applying CF seemed like the most compelling choice since this inherently combines content-based and collaborative filtering into the recommendation equation. In practice, my implementation of clustering items fell short after attempts with both k-means and topic modeling using latent dirichlet allocation (see LDA section).

As a result, I most heavily pursued Option 4 of clustering users followed by collaborative filtering. For the selected user, this would effectively filter his/her similar users in the first stage. The CF model implemented earlier would then be run on this filtered dataset.

K-means Clustering

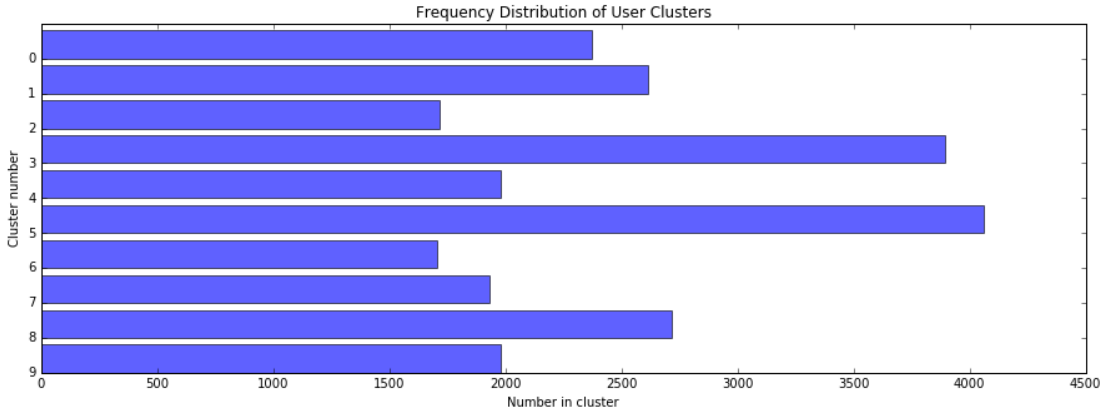
K-means clustering is one of the most commonly used clustering methods. A predetermined number of center points are set in random locations along n-feature space. Each user is then assigned to one of these points based on Euclidean distance, thereby updating the center points recursively until a steady state is achieved.

PySpark's MLlib package includes a distributed k-means clustering implementation. While the dataframe axis cannot be specified as a parameter, transposing it can achieve the same effect of clustering on columns (jokes) or rows (users). As per my Week 10 mini-project, I found that setting the maxIterations parameter equal to 10 yielded optimal results. The number of k center points depended more on the axis of interest.

Initially, I tried clustering jokes based on feature vectors but the length of these vectors along with the curse of dimensionality caused both runtime errors and/or ambiguous cluster centers (often zero arrays). Clustering jokes based on ratings yielded more efficient runtime but also undesirable results. With k=5, the frequency distribution of 100 jokes often ended very lopsided, such as below. Varying the value of *k* and other parameters also produced imbalanced cluster counts.

```
Counter({2: 63, 0: 34, 1: 1, 3: 1, 4: 1})
```

K-means clustering on users was much more effective. With k=10, the frequency distribution of clusters was quite balanced and provided a fair tradeoff of efficiency and granularity. The root sum of squares distance was 130.33.

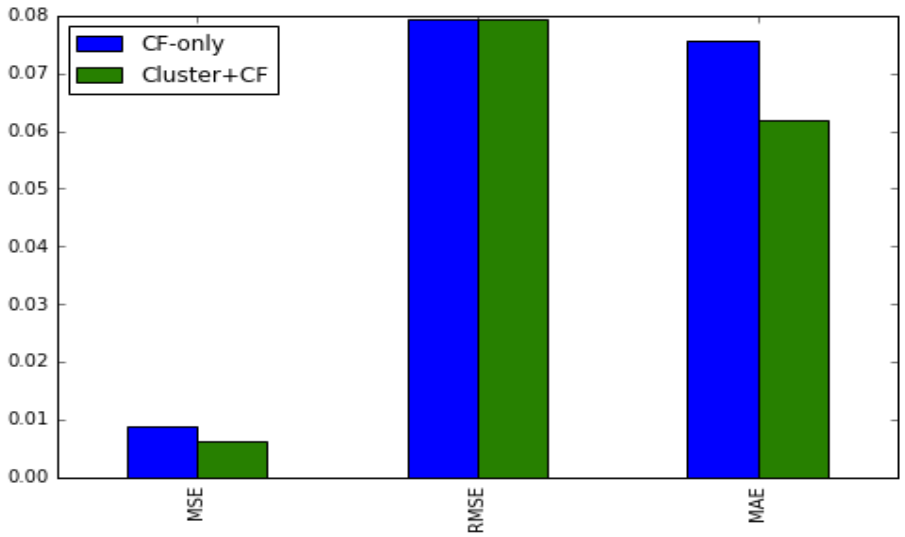


Latent Dirichlet Allocation (LDA)

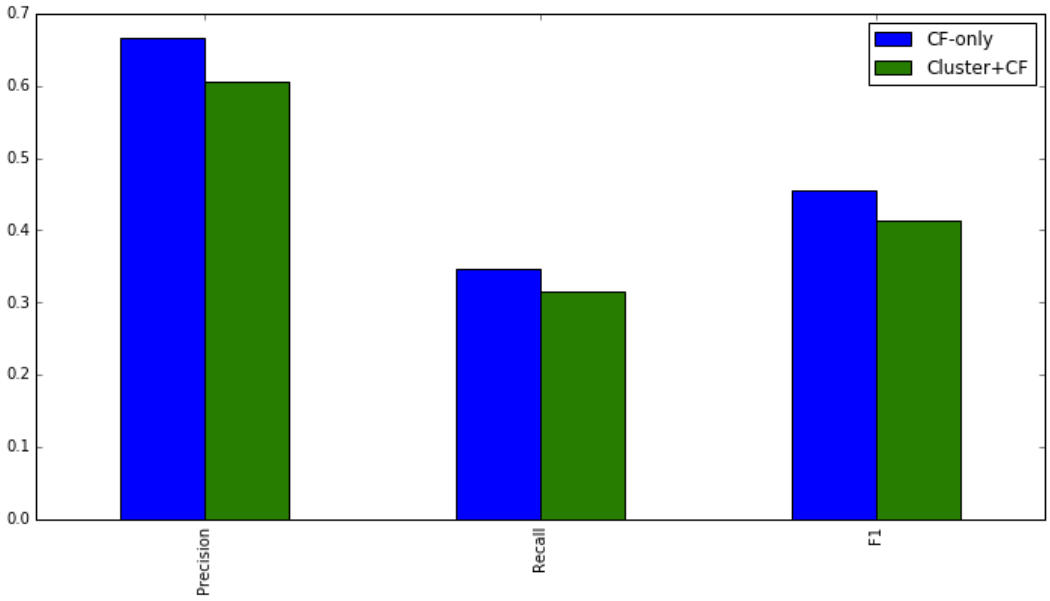
LDA is another clustering algorithm that extracts topic features, rather than the word features of TF.IDF from a corpus of documents. This would seemingly be helpful for our analysis as topics may be more informative than words for clustering jokes and identifying common features. Unfortunately, I ran into severe runtime and memory errors while executing the PySpark LDA implemementation. A separate attempt was made using SciPy's sparse matrices, but that data structure was not compatible with PySpark. Some further notes are in the notebook appendix.

Two-Stage Clustering and CF Results

K-means clustering on the user was the selected clustering method for the two-stage approach going forward. After selecting the predicted cluster for the specified user, the CF model is re-trained using only users and ratings from that same cluster. The rest of the ALS model, however, remained the same. Likewise, the train and test set approach for error measurement was consistent to allow for direct comparison with the CF-only model. The results were similar but slightly favored the combined model overall.



The recommendation accuracy of the combined model was also compared to the CF-only model using the same precision, recall, and F1 score process as the CF versus CB comparison. Interestingly here, the results favored the CF-only model, which was surprising given the perceived error advantages of the combined approach. My guess is that both stages are effectively reducing the user matrix and the clustering pass may be overdoing it. Given more time, I would look at the effects of increasing the number of center points in the k-mean model.



Computational Performance

While the Jester dataset is not the largest and much longer than wider, PySpark and distributed datasets really improved the efficiency of this project, particularly:

TF.IDF feature extraction

It is difficult to quantify the exact time saving of the PySpark TF.IDF implementation since I did not code a manual algorithm. PySpark decoded the corpus of 100 jokes in an average of 233ms. A non-hashing approach would conceivably take *O(NW)* units where N is the number of items and W the number of words per item.

Collaborative filtering / ALS algorithm

PySpark executed the training algorithm in an average of 13.9secs and the CF recommendation in an average of 329ms. In comparison, the Week 11 textbook exercises took us through a few iterations of distance measurement and re-normalizing a 3x8 user matrix. These primitive steps alone took about 100ms combined (calculated through R), which would be significant if scaled to a matrix factorization model as large (or larger!) than this one here.

Content-based user profile and recommendations.

The user profile algorithm for the CB system was one of the most expensive as it involves matrix multiplication and aggregation of the TF.IDF sparse vectors. My PySpark function using its linear algebra computations accomplished this in an average of 1.54secs. The content-based recommendations used similar linear algebra techniques to merge user and item profiles together, which PySpark executed in an average of 188ms. Although I do not have a direct baseline comparison, I would guess the distributed dataset savings are quite large here.

Two-stage clustering and CF

The two-stage approach is actually computationally expensive. Although the k-means clustering of users only needs to be done once offline, it took an average of 1m48s. Then for each specified user, the ALS training model must be re-run to filter for the predicted cluster only. This is in contrast to the CF-only approach where the training model only needs to be run once at the beginning. The combination recommendation then took an average of 244ms.

Limitations

The Jester dataset is really devoid of any details about the users themselves. Presumably they are from a common population group but it is unknown what is the expected similarity we might expect. This also makes it hard to evaluate whether our clustering results make logical sense when determining user groups.

The number of jokes is also limited to 100. If you scroll through the actual joke text, you can pick out some themes (e.g. relationship, cultural, puns) but a larger corpus with the potential to extract more common features would probably aid both CF and CB methods.

My short experience using PySpark is also a limitation, particularly in terms of runtime efficiency, which I'm sure could be improved and take advantage of partitioned datasets even more. I did find several instances where I almost preferred to execute the code in pandas and scikit-learn, but defaulted to PySpark for the integrity of the project. PySpark is also still missing some libraries and data structures that would have been helpful, most notably reading scipy csc_matrix as a sparse vector for LDA analysis.

Conclusions

The results of this project show that humor in the form of written jokes is not an entirely unquantifiable concept. Using a database of nearly 25,000 users, there was evidence that similar users and similar jokes can be identified based on ratings and word features alone or in combination. In the testing models, respectable mean absolute errors around 7% and precision of approximately 70%, when picking out the top half of rated jokes, were found. A content-based system surprisingly performed better than a collaborative filtering method while a two-stage clustering

and CF model did not provide significant accuracy or computational advantages over the single CF approach. It would be worthwhile to extend this project to an even larger dataset, particularly on the number of jokes dimension. Analyzing the joke text using n-grams or word2vec algorithms may provide even stronger feature extractions to improve the content-based models. Similarly, a deeper dive into topic analysis and LDA could give a two-stage recommendation model a stronger performance boost than the k-means clustering approach pursued here.

References

1. McMorris RF, Boothroyd RA, Pietrangelo DJ. Humor in Educational Testing: A Review and Discussion. Appl Meas Educ. 1997 Jul 1;10(3):269–97.
2. Carretero-Dios H, Pérez C, Buela-Casal G. Content Validity and Metric Properties of a Pool of Items Developed to Assess Humor Appreciation. Span J Psychol. 2009 Jul 1;12(2):773.
3. Ephratt M. What's in a Joke? In: Golumbic MC, editor. Advances in Artificial Intelligence [Internet]. Springer New York; 1990 [cited 2015 Dec 19]. p. 43–74. Available from: http://link.springer.com/chapter/10.1007/978-1-4613-9052-7_3
4. Eakin E. If It's Funny, You Laugh, But Why? The New York Times [Internet]. 2000 Dec 9 [cited 2015 Dec 19]; Available from: http://www.nytimes.com/2000/12/09/arts/if-it-s-funny-you-laugh-but-why.html
5. Goldberg K, Roeder T, Gupta D, Perkins C. Eigentaste: A constant time collaborative filtering algorithm. Inf Retr. 2001;4(2):133–51.
6. Rajaraman A, Ullman JD, Ullman JD, Ullman JD. Mining of massive datasets [Internet]. Cambridge University Press Cambridge; 2012 [cited 2015 Dec 20]. Available from: http://www.langtoninfo.com/web_content/9781107015357_frontmatter.pdf
7. Herlocker JL, Konstan JA, Terveen LG, Riedl JT. Evaluating collaborative filtering recommender systems. ACM Trans Inf Syst. 2004;22:5–53.
8. Shani G, Gunawardana A. Evaluating recommendation systems. In: Recommender systems handbook [Internet]. Springer; 2011 [cited 2015 Dec 17]. p. 257–97. Available from: http://link.springer.com/10.1007/978-0-387-85820-3_8
9. O'Connor M, Herlocker J. Clustering items for collaborative filtering. In: Proceedings of the ACM SIGIR workshop on recommender systems [Internet]. Citeseer; 1999 [cited 2015 Dec 20]. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.46.6325&rep=rep1&type=pdf
10. Pham MC, Cao Y, Klamma R, Jarke M. A Clustering Approach for Collaborative Filtering Recommendation Using Social Network Analysis. J UCS. 2011;17(4):583–604.