CMPE 256 PROJECT

Joke Recommendation System

Team Group - JRS

## By

## Amruta Dhondage

## Navneet Jain

## Surbhi Jain

Ch.1 Introduction

**Motivation:**

We started project topic search from the list of dataset options provided on canvas. We first had come up with 3 topics. 1> Delicious social bookmarking system. 2> Role of idioms in sentiment analysis 3> Joke Recommendation System by Jester. Due to small dataset we dropped second topic. Delicious dataset has social networking, bookmarking, and tagging information from a set of 2K users but due to lack of problem formulation we went for third topic. With Jock Recommendation system although we had small jokes dataset, we could get records for 25k users. This dataset made us think about other problems like creating joke categories and finding out users of certain joke category. This sounded fun working on hence we finally chose JRS.

We propose to build a joke recommendation system based on user’s preferences using the data obtained using UC Berkeley’s Jester Portal. Jester has a collection of hundreds of jokes within millions of ratings provided by hundreds and thousands of users. We aim to a model which understand user’s preference, likes/dislikes to recommend the most engaging jokes for the user. We further aim to analyze the categorization of jokes into multiple domains. For recommendation purpose, we will try to find the nearest domain and then predict the jokes from the closest domain for a given user.

Secondly, we can categorize the jokes into buckets and use the user ratings data to place every user in a particular category of jokes. Each user can like more than one category of jokes. By this we can find a target audience for a new joke in the market. The approach will be, place the new joke in one of the buckets and the send it to the users in that bucket.

**Objectives**:

1. Build a recommendation system which looks for user’s preferences

2. Perform categorization of jokes

3. Mapping the users to nearest category of jokes which user might be most

interested in.

4. Finding the target audience for each new joke in the market. Like this new jokes will only be sent to the audience who are most likely to read them in comparison to sending to every user.

Ch.2 System Design & Implementation details

**Algorithm considered**:

We focused on recommendation system and joke categorization. To get the best results we considered multiple algorithms.

**Joke Recommendation System:**

1. *Ranking Factorization*
2. *Rating Factorization*
3. *Item Based Factorization*

*Train all these three models to obtain the best model. All of these models take user-item rating matrix*

**Joke Categorization:**

*1. Gradient Boosting*

*2. Naive Bayes*

*3. SVM*

*4. Logistic Regression*

*5. Nearest Neighbors using Centroid*

*6. K-nearest Neighbors*

*7. Ensemble of all above classifiers*

**Joke-User Bucketing:**

*1. Nearest Neighbors*

*2. K-means clustering*

*3. Naive Bayes*

*4. Random Forest*

**Technologies & Tools**:

We were familiarize with python and jupyter notebook. Hence chose to start with it.

Along with these technologies, we used python’s libraries – scipy, numpy, panda, sklearn and Google News Vector Negative(word to vector generation, ref. https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit)

We used Github for our repository, regular commits and collaboration.

**System Design**:

Categorization System

Classification Analysis.

Jester Joke Text Dataset

Jokes with Categories Results.

A: Joke Categorization

Recommendation System

Recommendation Analysis.

Jester Joke User Rating Dataset

Joke Recommendation Results

B: Joke Recommendation

User Buckets for Jokes depends on categories

Bucketing System

Classification with Clustering Analysis.

Results from A and B

C: Joke-User Bucketing

System A, B and C shows the Joke Recommendation System working. Each system works on the algorithms mentioned in earlier sections. Data is preprocessed and collected into needed formats for each system. Results are compared and evaluated for accuracy.

Ch.3 Experiments

**Dataset:**

* Dataset is taken from Jester ( UC Berkley) <http://eigentaste.berkeley.edu/dataset/>
* jester\_dataset\_1\_1.zip: (3.9MB) Data from 24,983 users who have rated 36or more jokes, a matrix with dimensions 24983 X 101
* Ratings are real values ranging from -10.00 to +10.00 (the value "99"

corresponds to "null" = "not rated").

* The first column gives the number of jokes rated by that user. The next 100 columns give the ratings for jokes 01 - 100.
* The text of the jokes can be downloaded here:

[jester\_dataset\_1\_joke\_texts.zip](http://eigentaste.berkeley.edu/dataset/jester_dataset_1_joke_texts.zip) (92KB) / [jester\_dataset\_2.zip](http://eigentaste.berkeley.edu/dataset/jester_dataset_2.zip) (7.7MB)

**Data Processing**:

We worked on two datasets User-Joke ratings and Joke Text. Before feeling data into algorithm, we needed to process them in appropriate format.

1. User-Joke Rating:

From User-Joke rating dataset we formatted data into userid-jokeid-rating matrix. This dataset is then fed to recommendation system

1. Joke Text Processing:

Joke text dataset was raw text dataset. We processed it into jokeid and joketext structure. Joketext had many unwanted words and characters. We processed the text for following conditions to get the clean joke text.

|  |
| --- |
|  |
|  | joke = re.sub(r'([^**\.**\s\w]|\_)+', '', joke).replace(".", ". ") |
|  | joke = joke.replace('\r', '') |
|  | joke = joke.replace('\n', '') |
|  | joke = joke.replace('<br />', '') |
|  | joke = joke.replace('<p>', '') |
|  | joke = joke.replace('&quot;', '') |
|  | joke = joke.replace('&#039;', '') |

1. Joke-User Bucketing:

We needed to have categorization for joke and user likes and dislikes for specific category. Data is processed for having a userid-jokecategory-jokeid-rating structure.

**Methodology followed**:

**Recommendation System**:

1. Ranking Factorization:

A RankingFactorizationRecommender learns latent factors for each user and item and uses them to rank recommended items according to the likelihood of observing those (user, item) pairs. This model cannot be constructed directly. Instead,use [graphlab.recommender.ranking\_factorization\_recommender.create()](https://turi.com/products/create/docs/generated/graphlab.recommender.ranking_factorization_recommender.create.html#graphlab.recommender.ranking_factorization_recommender.create) to create an instance of this model. RankingFactorizationRecommender contains a number of options that tailor to a variety of datasets and evaluation metrics, making this one of the most powerful models in the GraphLab Create recommender toolkit.[ Ref. <https://turi.com/products/create/docs/generated/graphlab.recommender.ranking_factorization_recommender.RankingFactorizationRecommender.html>]

1. Rating Factorization:

Create a FactorizationRecommender that learns latent factors for each user and item and uses them to make rating predictions. This includes both standard matrix factorization as well as factorization machines models (in the situation where side data is available for users and/or items). [ Ref. <https://turi.com/products/create/docs/generated/graphlab.recommender.factorization_recommender.create.html>]

1. Item Based Factorization:

This model first computes the similarity between items using the observations of users who have interacted with both items. Given a similarity between item ii and jj, S(i,j)S(i,j), it scores an item jj for user uu using a weighted average of the user’s previous observations IuIu. [Ref. <https://turi.com/products/create/docs/generated/graphlab.recommender.item_similarity_recommender.ItemSimilarityRecommender.html>]

Train all these three models to obtain the best model. All of these models take user-item rating matrix

Steps:

1. Split the data in train and validation using Graphlab library

2. Define all these models (here in: recommendation\_modules method)

3. Train these models for various latent factors: Factors required in factorization

4. Optimizing for RMSE

5. Plot various models to obtain which model and factor performed the best.

**Joke Categorization**:

This is the Code for predicting the category of joke from the following categories:

1. Animals

2. Technology

3. Doctor

4. Man

5. Politics

6. Relationship

7. Religion

8. School

9. Food

10. Others

We have pre-labelled the jokes data (150 jokes) into the above mentioned classes. Following is the distribution of the categoies:

8 animal

7 doctor

8 food

1 joke\_category\_reduced

11 man

58 others

15 politics

13 relationship

1 religion

10 school

19 technology

Approach:

Features for Jokes:

1. Obtain key words for each joke

2. Obtain Glove vectors (similar to Word2Vec) from pre-trained based on Wikipedia data of 300 dimension for each word and then averaging out for the entire joke.

3. These averaged out 300 dimensional Glove vector for joke is used as feature for classification

Classes:

10 categories mentioned above.

Training:

The problem is posed as a multi-class classification problem with 10 classes. Since the data is of small size, only 10% of the jokes are used for testing while

remaining 90% jokes are used for training. The following models were used:

1. Gradient Boosting

2. Naive Bayes

3. SVM

4. Logistic Regression

5. Nearest Neighbors using Centroid

6. K-nearest Neighbors

7. Ensemble of all above classifiers

**Joke-User Bucketing:**

Approach:

1. Obtain the Joke categorized dataset
2. Retrieve jokes for the specified category
3. Classify/cluster user likes and dislikes for the specific jokes under specified category.
4. Show users liking the jokes belonging to specified category.

Training:

Dataset is divided into 90% training and 10% testing data.

1. User likes and dislikes are decided from ratings.
2. Model is trained for user likes and dislikes.
3. Classification in 0:dislike, 1 :like is done using algorithms mentioned.
4. From the results and categorization output, joke-category to user bucketing is done.

**Evaluation**:

1. *Name: Gradient Boosting*

*Accuracy score: 0.4*

1. *Name: Naive Bayes*

*Accuracy score: 0.466666666667*

1. *Name: SVM*

*Accuracy score: 0.533333333333*

1. *Name: Logistic Regression*

*Accuracy score: 0.533333333333*

1. *Name: Nearest Neighbors using Centroid*

*Accuracy score: 0.533333333333*

1. *Name: K-nearest Neighbors*

*Accuracy score: 0.466666666667*

1. *Name: Ensemble*

*Accuracy score: 0.466666666667*

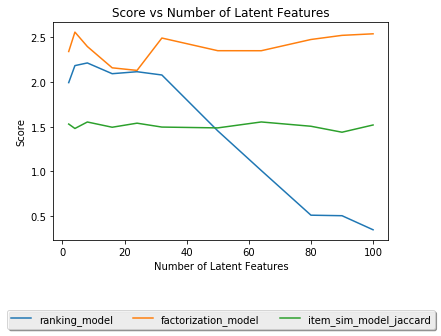
Even though Nearest Neighbors and LR gives best performance, however they often bias towards other class. On qualitative analysis, Naive Bayes, Gradient boosting does better generally. KNN does better when their are enough samples for a category, however it is biased towards other class which is a popular class making the accuracy to be high.

|  |
| --- |
| On validation: |
|  |
| *1. Ranking Factorization: Num Factors: 100 Score: 0.34862012987* |
| *2. Rating Factorization: Num Factors: 90 Score: 2.51836444805* |
| *3. Item Based Factorization: Num Factors: 90 Score: 1.43739853896* |

|  |
| --- |
|  |

**Analysis Results**:





Ch.4 Discussion & Conclusions Decisions made

* Difficulties faced
* Things that worked
* Things that didn’t work well
* Conclusion

Ch.5 Project Plan

|  |  |  |
| --- | --- | --- |
| Task | Assignee | Justification (if any) |
| Project Topic/ Dataset discussion | All |  |
| Project analysis | All |  |
| Joke Categorization | ? Surbhi |  |
| Joke recommendation | Surbhi |  |
| Joke-User bucketing | Amruta | Working |
| Project report/presentation | All |  |