**Data Mining Final Project**

**By**

**Joseph Barbati, Joseph Lee, Alvince Pongos**

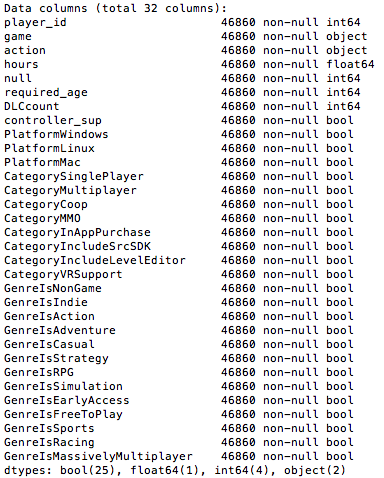
* **Motivation and contributions: motivate the problem that you are investigating and summarize your goals and contributions**
  + As all of the group members are dedicated gamers, we struggle to find new games that suit our taste. Therefore, we decided to use the dataset from Kaggle on Steam Video Games and built a recommendation system based on users’ gaming history.
  + Our goal is to create recommendation system that returns the top 10 games given an input of games that the user likes

* **Related work/methods: review existing work/methods that may be applicable for your project**
  + There exists related python libraries such as SciKit’s Surprise written by Nicholas Hug that performs several recommendation algorithms. We used this library to compare our code against their benchmarked algorithms.

* **Approach and methodology: describe your approach/solution including any algorithmic developments and/or prototype implementation and/or experimental methodology**
  + Our group created a recommendation system using Jaccard similarity with a rating threshold. In our recommendation system, we found frequent itemsets of all of the games with support of 5. We chose only items that met an 80th percentile threshold in hours played. We used these frequent itemsets as our base recommendation set.
  + In order to increase accuracy with our recommendation system, we added weights to our recommended games. The weights were determined by matching genre and similarity of title. If the title of a recommended game was similar to that of a game on the input list, the base weight (1) would be multiplied by 3. Furthermore, if the recommended game had matching genres, we multiplied the weight again by 2 for every matching genre.
  + Our system initially was executing at O(NMKS) time. (We said O(NMK) in our presentation, this was a mistake.)
    - * N = length of input list
      * M = number of subsets in input list
      * K = number of frequent itemsets
      * S = O(2N) -- time to build subset list
    - In order to make it run faster, we tried making the slowest part a bit faster.
      * We tried to optimize the creation of the subset list. We did this by adding a check to see if the current subset we are creating is even in the frequent itemsets we found. If not, we return from the recursive call in order to not do useless work.
      * Effectiveness? A length of 12 games took minutes before, now the same list takes seconds.
  + Using Scikit Surprise, we first had to transform our data into a format that the algorithm would accept (User, GameID, Rating). We normalized the data by mapping Hours Played to its respective percentile to get an implicit rating. We then used three fold cross validation on our entire dataset and performed Single Value Decomposition (SVD), KNN Basic, KNN with Means, SVD++, Slope One, Negative Matrix Factorization, and CoClustering to get average Root Mean Square and Mean Absolute Error for each test.
  + In order to evaluate our recommendations, we created a test set which took the top four games from each user and randomly removes 1 - 3 of the games to see if our system recommends them back. Then we calculate precision and recall or each recommendation list.

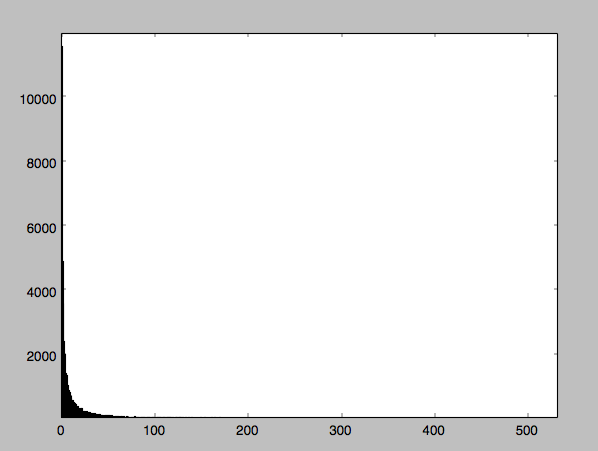
* **Evaluation and results: describe your datasets, evaluation metrics, and experiment parameters, present and discuss your results.**
  + Steam video games data set has 4 attributes
    - A unique player-id, game, action, and hours played.
  + Game features dataset has 73 attributes, we only care about the 14 that include the name of the game and any genre information.
  + We combined the two data sets in order to input genre data for each game listed in steam video games.
  + In order to evaluate our recommendation system, we used precision and recall.
  + In order to use precision and recall, we created a test set from our steam video games dataset where we randomly ‘removed’ some games from the list. By ‘remove’ we mean add another attribute, 1 to include it in list into recommendation system and 0 to ignore. Then we see how many we will correctly recommend back and perform precision and recall calculations for each call to our recommendation system.
  + Precision is calculated as correctly recommended games / all recommended games
  + Recall is calculated as correctly recommended games / games that should have been recommended (i.e. games we set to ‘ignore’).
  + We ran our recommendation system on over 11,000 users and got the following precision and recall:
    - Precision (avg per user): 0.014559
    - Recall (avg per user): 0.20544
  + Our precision and recall are poor, but it does not take into account the fact that we recommend games that, should a user play, they actually would like and rate highly.

**Key Statistics and Visualization**



*Figure shows meta information about our dataset.*

***Histogram of Hours Played***



*Figure shows histogram of hours played per game for all players (autobin method takes the maximum of sturges and Freedman Diaconis). Zoomed in to see long-tail feature.*

The hours played is an important metric because we are using it as a proxy to determine how good a game is. The more hours a game is played, the better we assume it to be. To get a better sense of how we should turn Hours Played into ratings we calculated these key statistics.

**Key Statistics of Hours Played**

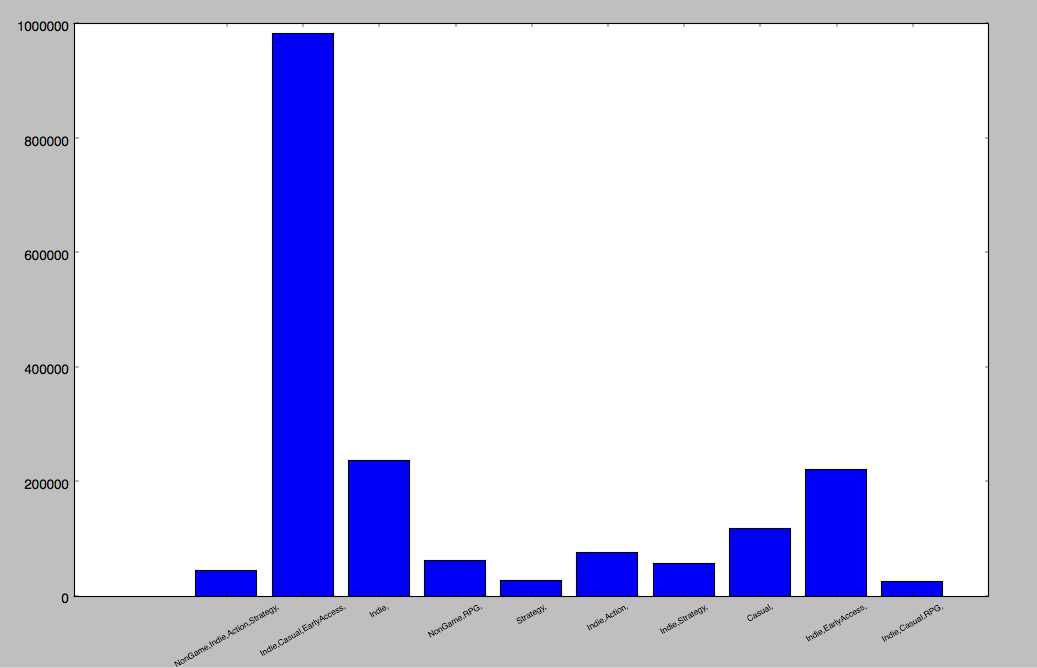
|  |  |
| --- | --- |
| **Statistic** | **Value (hours)** |
| Min | 0.1 |
| Max | 11,540 |
| Median | 3.9 |
| Mode | 0.2 (2031) |
| 75th Percentile | 16.3 |
| 25th Percentile | 1.0 |

*Figure shows key statistics of the hours attribute*

**Key Statistics of Genres**

|  |  |
| --- | --- |
| **Statistic** | **Value** |
| Unique Comb. of Genres | 247 |
| Genre Played Most (Hrs) | “Indie,Casual,EarlyAccess” (982971) |
| Genre Played Most(Count) | “Indie” 6281 |

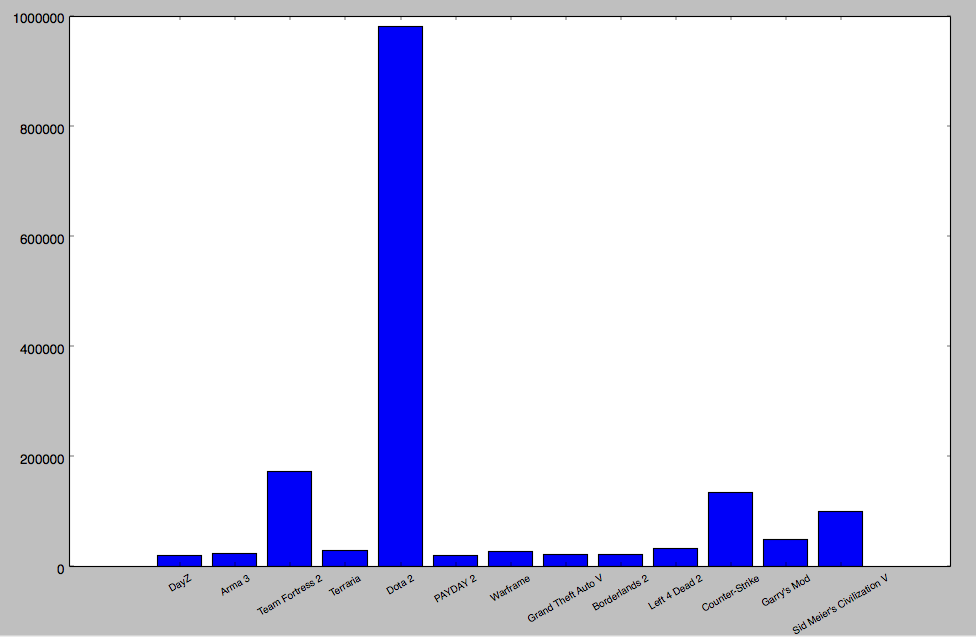
*Figure shows key statistics of Genres played Most (Hrs) in entire dataset.*



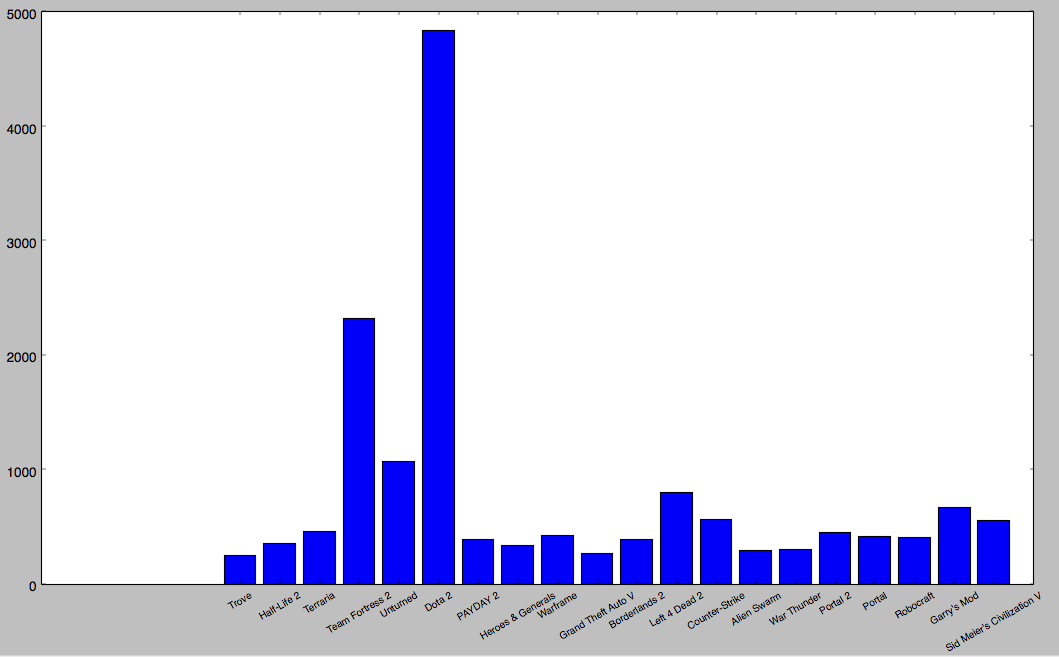
*Figure shows barchart of the top ten most played Genres. Many of the Genres include Indie,RPG,Casual,Action, and Early Access.*

**Key Statistics of Titles**

|  |  |
| --- | --- |
| **Statistic** | **Value** |
| Title Played Most (Hrs) | Dota 2 (4841) |
| Title Played Most(Count) | Dota 2 (981795) |



*Figure Shows the top 13 Titles played (hours).*



*Figure shows top 20 Titles played (count)*

Some of the most popular games played by count or by hours include Dota 2, Team Fortress 2, and Counter Strike. Since these are so popular, we expect our recommendation system will likely recommend these three games often.

**Average Hours Played Percentile Chart**

|  |  |
| --- | --- |
| **Percentile** | **Title, HoursPlayed** |
| 5.0%': | ["Tom Clancy's Ghost Recon", 0.25], |
| '10.0% | ['Psycho Starship Rampage', 0.4] |
| '15.0%' | ['MirrorMoon EP', 0.6] |
| '20.0%' | ['Retro/Grade', 0.95], |
| '25.0%' | ['HeXen II', 1.3333333333333333] |
| '30.0%' | ['Zeno Clash', 1.675] |
| '35.0%' | ['Final Dusk', 2.0] |
| '40.0%' | ['Vox', 2.4] |
| '45.0%' | ['Waveform', 2.85] |
| '50.0%' | ['8BitBoy', 3.4] |
| '55.0%' | ['Waking Mars', 4.14] |
| 60.0%' | ['WRC 4 FIA WORLD RALLY CHAMPIONSHIP', 5.04] |
| '65.0%' | ['Call of Juarez', 6.0] |
| '70.0%' | ['Spectraball', 7.5] |
| '75.0%' | ['The Raven - Legacy of a Master Thief', 9.475] |
| '80.0%' | ['Bound By Flame', 12.24] |
| 85.0%' | ['Dead or Alive 5 Last Round', 16.3] |
| '90.0%' | ['Sleeping Dogs', 23.12] |
| '95.0%' | ['Victoria II', 41.07] |
| '99.0+%' | ['Eastside Hockey Manager', 1295.0] |

*Figure shows percentiles with respect to hours played. We use these intervals to implicitly determine ratings of games.*

**SciKit Surprise Recommendation Library Errors**

|  |  |  |
| --- | --- | --- |
| **Method** | **Root Mean Square Error** | **Mean Absolute Error** |
| SVD | 0.3407 | 0.2858 |
| KNNBasic | 0.3084 | 0.2621 |
| KNNWithMeans | 0.3149 | 0.2592 |
| SVDpp | 0.3015 | 0.2563 |
| SlopeOne | 0.3272 | 0.2716 |
| NMF | 0.3191 | 0.2645 |
| CoClustering | 0.5057 | 0.4228 |

*Figure showing the Root Mean Square Error and the Mean Absolute Error for many different recommendation methods on our dataset. Implementation uses 3 fold cross-validation. Single Value Decomposition gives the least RMSE and MAE. CoClustering gives the greatest errors.*

Documentation for each of these methods can be found here:

<http://surprise.readthedocs.io/en/v1.0.2/prediction_algorithms_package.html>

Documentation to for how to show recommendation errors can be found here:

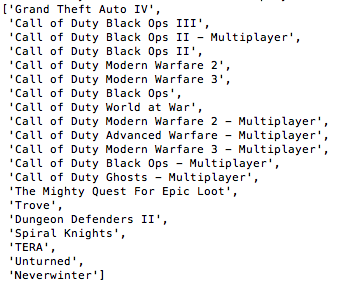
<http://nbviewer.jupyter.org/github/NicolasHug/Surprise/blob/master/examples/notebooks/KNNBasic_analysis.ipynb>

**Sample Input and Output of Our Recommendation System**

Sample Use:



Sample Output:



Note: the order in which you see the games matters, the first game is the ‘most recommended’ meaning it had the most weight (determined by matching genres and titles).

**Conclusion and future work: a) discuss what you have learned through the project and what concepts and techniques you learned in class are used in the project; and b) discuss potential extensions and future work.**

1. Below is a list of the concepts we practiced during the implementation of our recommendation system.
   1. Combining different datasets. Feature Subset Reduction.
   2. Normalizing ratings implicitly with percentiles.
   3. Frequent Itemset mining.
   4. Direct Hash Pruning.
   5. Jaccard similarity with threshold.
   6. Data preprocessing and visualization.
   7. Evaluation metrics such as RMSE, MAE, Precision, and Recall
2. Potential future work we could do is to include what we predict a user would rate a game. We would want to do this in order to have a more comparable metric to our other recommendation systems we used through sci-kit surprise.While our recommendation system is not slow for small N (input list length), if it were increased to 20, 30, or even higher, there would be a sharp increase in time to execute. Because of this, more potential future work would be optimizing our algorithm. In addition, we could also include user-user collaborative filtering. Right now, our system uses global frequent itemsets to recommend games.