Prediction of a Hit TV Show

By Team 6 -
Yao-Chia Yang,
Howard Chien,
Miaoyan Zhang

Preface

Currently, there are several methods to predict whether movies will be popular or not. Although a large number of research papers are focused on exploring the factors affecting movie ratings, very few studies have looked into TV series. While movies and TV series have similarities, TV series are different from movies in many aspects that worth investigating. In addition, most of them only collect data from one or two platforms. The project is going to collect data from different platforms and find a better method to predict the rating on IMDb.

Acknowledgements

First of all, we would like to show our gratitude towards Dr. Ming-Hwa Wang for his guidance and support. We are also immensely grateful to authors of the paper in the bibliography. Last but not least, we thank Santa Clara University for providing resources to help us complete the project.

Table of Contents

List of Tables

List of Figures

Abstract

The prediction of a hit TV series can help producers and investors decide whether a new TV is worth investing. Also, it can help directors make more popular TV series and get more profits from it.

In the project proposal, we develop a method to predict the rating of TV series on IMDb by a kernel-based approach. We use different data source from Twitter, TVMAZE and IMDb to make more precisely prediction.

The results indicate that using SVD will generate closer prediction compared to real IMDb rating. Also, we can confirm that some features have little affection on our predictive result, which can be eliminated.

I. Introduction

1.1 Objective

Our objective is to build a systematic method to predict the rating of TV series on IMDB.

1.2 What is the problem

It's hard to decide whether investors should invest their money in new TV series or not. Many factors will affect the success of TV series, such as directors, actors, writers, etc. Therefore, the problem of how to make a prediction of a hit TV shows comes to our attention. We not only predict the rating of TV series, but also determine what factors affect the prediction most.

1.3 Why this is a project related this class

In this project, we mainly use the kernel-based approach and SVD to make a prediction of a hit TV shows. Also, we combine other data mining methods, such as sentiment analysis.

1.4 Why other approach is no good

Most of other approaches about predicting of movies or TV series are only focus on one factor, such as TV series scripts or social media feedback. We collect different kinds of data to make the prediction, like followers count from Twitter (TV show-based), platform (release-based), audience reviews toward actors(audience-based), etc.

1.5 Why you think your approach is better

By using comprehensive data, we can get more precisely prediction. We also combine the advantage of kernel-based approach and sentiment analysis to generate our result.

1.6 Statement of the problem

Our goal is to predict the rating of TV shows on IMDb to determine whether it will be a hit TV series. Also, we determine some important factors which can help investors make a right decision, such as the following:

1) What are the feature weights?
2) How is the prediction accuracy when only using content-based(TV show-based) data?
3) Will adding released based data and audience-based data for input improve the prediction accuracy?

1.7 Area or scope of investigation

Our scope of investigation includes data mining and machine learning. We especially focus on kernel-based approach to implement our prediction. Also, we use SVD to do the dimension reduction for our normalized data, which can help us reduce the huge amount of TV series data.

II. Theoretical Bases and Literature Review

2.1 Definition of the problem

As mentioned in section 1, many factors may affect the TV series' rating score. We can find some paper in finding out main factors in movie, but find few paper in TV series. TV series share some similarities with movies, but have some differences with movies as well. So this paper invests on what factors affect the success of a TV series, and using these findings to predict whether a new TV series will be a hit.

2.2 Theoretical background of the problem

TV show rating prediction could be treated as part of the recommendation problem on the TV show. Content-based systems and collaborative filtering are two basic architectures for a recommendation system.

1. For a content-based systems, similarity of TV shows is determined by measuring the similarity in their features.
2. For an item-based collaborative filtering, ratings could be determined by ratings of those similar TV shows.

2.3 Related research to solve the problem

In the paper Assessing Box Office Performance Using Movie Scripts- A Kernel-Based Approach1, kernel-based approach is used to predict the BOX office of the movies based on the contents of movie script. The data source is simply movie script, but it looks into the data and extracts different levels of information. Since there are different levels of variables, intuitively we think that the importance of these variables should be different. Some of them will contribute more while some of them contribute less. By weighting these variables, they find the similar movies and use them to predict the BOX office performance. This approach is similar to item-based collaborative filtering.

For another project research, some use linear regression method as the way to train the predictive model. To improve the regression model, they have also fitted a weighted linear model and a reduced linear model by principal component analysis.

2.4 Advantage/disadvantage of those research

Advantage:
In the kernel-based approach, they could only use the movie script in the pre-production phase to approximately predict the box office performance.

Disadvantage:
In the data extraction phase, some of the data should be human input, and it is hardly conducted by computers. This strongly limits the availability of this prediction system.

2.5 Your solution to solve this problem

1) Use kernel-based approach to find out the weight of each factor: title, director, actor/actress, writer, genre, synopsis, duration, release platform.
2) Sentiment analysis of reviews from IMDB for each director/producer, actor/actress, writer.

2.6 Where your solution different from others

1) We collect data from wider range. Use number of followers of each actor/actress, director and writer on Twitter, and collect review data from IMDB.
2) We analyze more factors from different dimensions: audience-based data, release-based data and TV series-based data.
3) We combine the kernel-based approach and sentiment analysis.

2.7 Why your solution is better

1) We try to analyze using larger dataset. Larger data source may improve the predicting accuracy.
2) We try to combine different method which may improve the predicting accuracy.

III. Hypothesis

3.1 Hypothesis

1. Content-based (TV series-based) data are the main factors to influence the accuracy of the prediction of ratings.
2. There will mainly be three kinds of data sources that correspond to different phases for a TV series, content-based data for production phase, released-based data for pre-release phase, and audience-based data for post-release phase. As we got more types of data sources, we could predict the ratings more and more accurately.

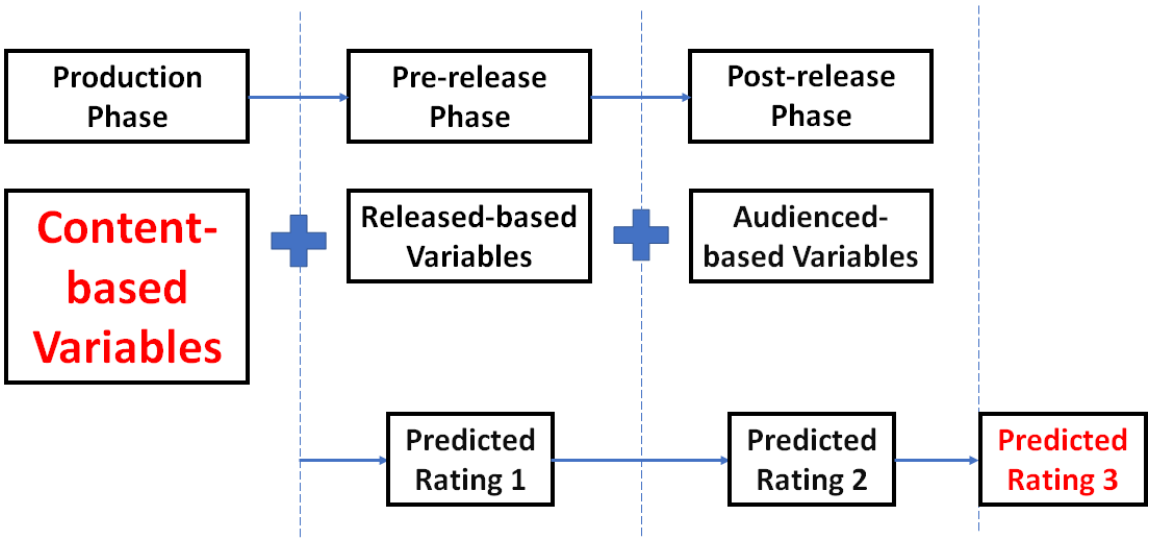

Figure 1. Hypothesis

IV. Methodology

4.1 How to generate/collect input data

There are three groups of data, content-based data, release-based data, and audience-based data.

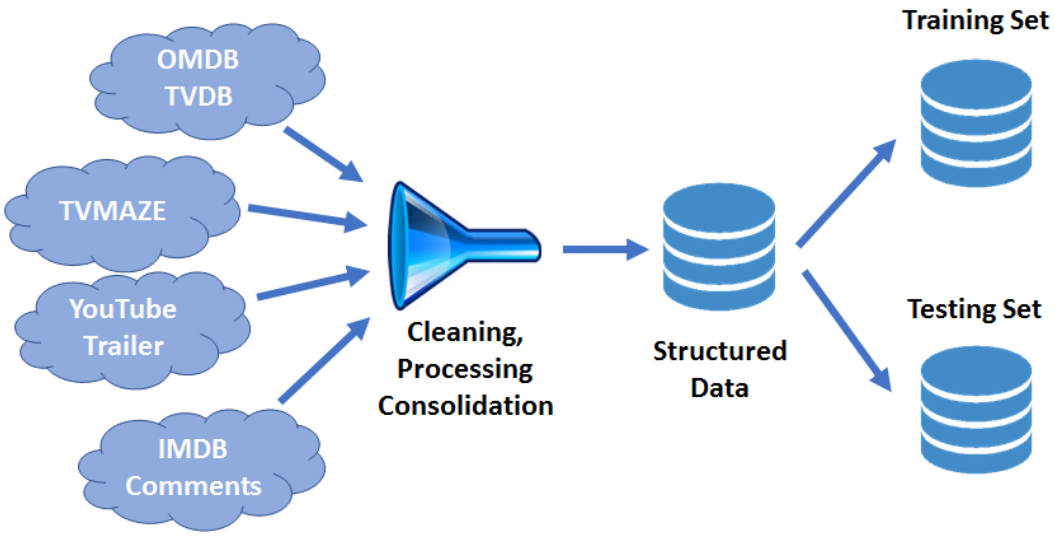

Figure 2. Collect Data Flow

Content-based data:

Firstly, we use the open datasets of TVMAZE to get all the instances, which include features like 'title', 'genre', 'duration' and 'synopsis'. We then use the imdbid to get the corresponding director/actors/actresses information from OMDb and TVDB.

Release-based data:

The release-based data, like release platform and release date, could be obtained from open datasets of TVMAZE.

Audience-based data:

Reviews data: Use the IMDB RESTful JSON API to mine out the popularity of each director/producer, writer, and actor/actress.

4.2 How to solve the problem

To predict the TV show ratings, our first intuition was to use item-based collaborative filtering or kernel-based approach. That is, the more similar the TV shows they are, the more likely they will get similar rating of the corresponding TV show. We then use the similar TV shows to predict the rating of a testing sample and set the rating as our objective/response variable for the testing sample.

4.3 Algorithm design

The first step is to clean the datasets and to organize the instances as the standard formats as we want. Each instance will have several feature variables and they will be assigned a score to indicate how do they perform in that concept feature. Here are our proposed features for a TV show. All of these data are then normalized into the integer rank of 0 to 1.

- TV-show based
    - Director's twitter followers
    - Actors' twitter followers
    - Writer' twitter followers
    - Genre(eight kinds of genre)
    - Synopsis
    - Duration
- Released-based
    - Release-date
    - Platform
- Audience-based
    - Review content
        - Actor/actress’s positive & negative reviews
        - Writer’s positive & negative reviews
        - Director’s positive & negative reviews

For review content, we use sentiment analysis to analyze the popularity of each actor/actress, director, writer. In the review analysis, we only use the review data from IMDb. Iterate through each TV's reviews to get each celebrity's review points. If one review is disliked by over 50%, discard that review for the review may be a fake one. If one sentence mentioned the name of celebrity in that TV, classify whether the sentence is positive, neutral or negative:

when positive: add 2 * (# of likes) points to the celebrity's positive review points

when negative: add 1 * (# of likes) points to the celebrity's positive review points

when negative: add 2 * (# of likes) points to the celebrity's negative review points.

Then we have each celebrity's review points.

For the sentiment analysis part, each review can be split by [.!?], and each sentence can be represented as a bag of words; with stop words removed. If the sentence contains the name of a celebrity extracted from the Stars attribute from the OMDb API. Stop words are words like "at", "a", "the". Then use Apache OpenNLP to train classifiers for positive, neutral, negative sentiments. We choose the MaxEnt algorithm which combines the Naive Bayes and Multinomial Logistic Regression.

We sampled 0.1% records from the logs of analyzing the review data, and checked manually, the accurate rates are shown as Table 1(Details can be referred in the appendix), so it is OK to be the input of prediction model.

	Actual #	Classified as 0 #	Classified as 1 #	Classified as 2 #	accuracy
0 - negative	34	25	8	1	73.53%
1- neutral	105	20	78	7	74.29%
2 - positive	135	19	25	91	67%

Table 1. Sentiment Analysis Accuracy

The second step, we use cross validation to train or test our collected data. Each time divide our samples into two sets, 80% for training and 20% for testing. Training set is used to build up our predictive model, while the testing set is used to validate our predictive model. In the predictive model, there are weighting parameters for feature variables and these parameters are used to characterize their contributions to the rating. Initially they would be treated equally to 1, and then minimize the error function by Nelder-Mead method or possible gradient descent algorithm as the optimization problem.

As the weighting parameters are determined, the predictive model is constructed and we could predict the rating of a sample in the testing set. Finally, we could compare our predicted results with those actual results.

4.4 Language used

We use JAVA to implement the algorithms, from extracting features from raw datasets, calibrating and optimizing the parameters, predicting the ratings, to comparing the error between predicted ratings and actual ratings.

4.5 Tools used

Java Ejml library for SVD:
http://ejml.org/wiki/index.php?title=Main_Page
Java Jama library for Multiple Linear Regression(Testing purpose):
http://math.nist.gov/javanumerics/jama/

4.6 How to generate output

The kernel-based method use the distance metric to determine the similarity between a testing sample and each observation in the training samples. Before training, we could assume each variable with equal contributions to the results, and use the training samples to calibrate the weighting parameter to find out the relative contribution to the ratings. This could be

accomplished by minimizing the error between actual and predicted rating.

After finding out the weighting parameters, we then need to find out a tuning parameter which determine the contribution of ratings from each TV show in the training sets. So far, we could use the tuning parameter such that there will be 10 most similar TV shows which dominates the predicted rating.

Based on all the feature variables and the calibrated variables, we could predict the rating of a TV show in the testing set.

4.7 How to test against hypothesis

For hypothesis one, we could use the calibrated weighting parameters to learn the importance of each variable. The more important the variable is, the larger the weighting parameter is. We could validate our hypothesis by looking into the top 5 weighting parameters.

For hypothesis two, as we have variables from three kinds of data sources. we could exclude different sets of variables and use the remaining features to train and calibrate our weighting parameters. We could compare the error between the predicted ratings and the actual ratings correspondingly.

V. Implementation

5.1 Code (refer programming requirements)

The entire set of source code pertaining to our project can be found at the appendix section of this document .

5.2 Design Document and Flowchart

A. Data acquisition ->
    a. TV show raw data
    b. Review/Comment -> NLP
B. Data Cleansing -> Data Normalization ->
C. Singular Value Decomposition -> Optimise Feature Weighting -> Predict the rating based on similarity

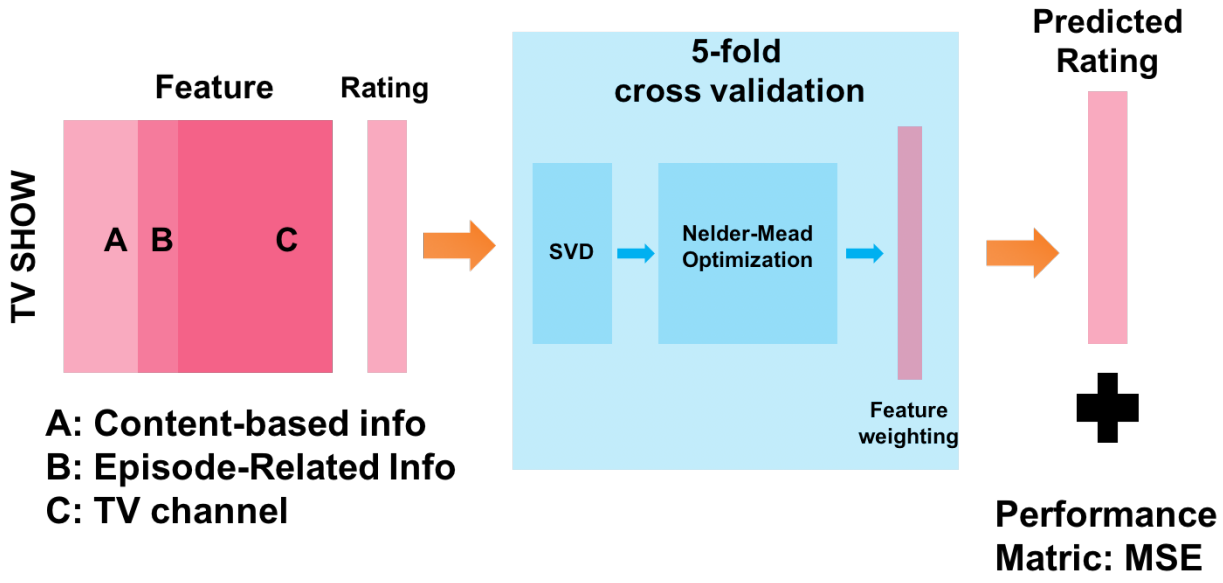

Figure 3. Implementation Flowchart

VI. Data Analysis and Discussion

6.1 Output Generation

To predict the ratings and evaluate our kernel-based models, we used 5-fold cross validation. We divided the entire dataset into 5 equal-length subsets. Each time we held one of the five as the test set, and trained on the other four as a whole. For each training set, we built a kernel-based model (feature weighting parameters set), and made predictions on the test set to obtain the mean squared error (MSE).

> *Mean squared error (MSE): Measures the average magnitude of errors in the predicted values. That is, the average distance of a data point from the fitted line. Being a quadratic measure MSE is most affected by large errors, thus useful for when large errors are especially undesirable.*

The following section aims to present an overview of the experiment results, demonstrating the prediction performance over the TV show dataset, consisting of 220 TV shows. As previously stated, all results from the experiment are based on a setup of 5-fold cross-validation combined with weighted search of model parameters, in order to optimize the parameters of each algorithm to better reflect its performance.

In order to first and foremost establish a better understanding of the predicted TV show data, a scatter plot over the predicted values is illustrated in Figure 4. Furthermore, the plot is followed by several box plots for each performance metric (i.e. MSE, Mis-classification rate).

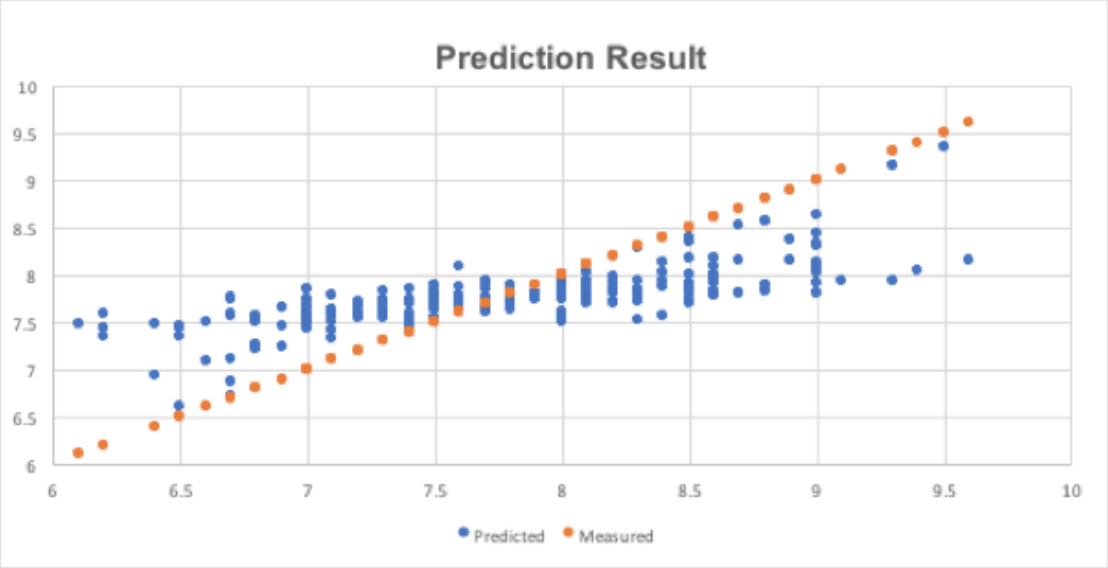

Figure 4. Prediction Ratings v.s. Actual Ratings

Figure 4. above illustrates the full dataset of 220 TV shows predicted by our algorithm using the same nested cross-validation as the rest of this experiment. It should therefore be noted that all of the data points in the illustration are predicted through a combination of 5 splits of training and test data, using the all same partitioning of data for both algorithms. Thereby, the algorithm parameters chosen for each iteration of the outer cross-validation loop, e.g. model assessment, might differ slightly from iteration to iteration as the partitioning of training data to some degree will vary.

6.2 Output Analysis

A = TV Show-based attribute + Review-based attribute

B = Release-based attribute

C = Channel-based attribute

As Table 2 shows, the full kernel-based model gave an error estimate of 0.517. It is still necessary to dig down to see the parameters we get.

Data Source	5-fold MSE
A ->	0.696
B ->	0.536
C ->	0.539
A + B ->	0.726
A + B + C ->	0.715
(A) -> SVD ->	0.547
(A + B) -> SVD ->	0.554
(A + B + C) -> SVD ->	0.517

Table 2. Predictive Performance (Mean Squared Error) for Different Data Sources

In Table 3, based upon the rating category table suggested by IMDB, we find out we could not predict the TV shows well in “VERY GOOD” and “GOOD” category. The reason will be discussed in the “Compare with the Hypothesis” part.

Original Rating	**Class**	**Counts**	**# Correctly Prediction**	**# Missed Prediction**	**Total # Miss**	**Total Miss Rate**
>= 8.5	Very good	44	1	43	75	33.93%
>= 7.0	Good	146	142	4		
< 7.0	Fair	30	2	28		

Table 3. Mis-Classification Rate

The following figures demonstrate the important features we find with different data sources. In the content-based only data source case in Figure 5, the genre related attributes dominate as we expected.

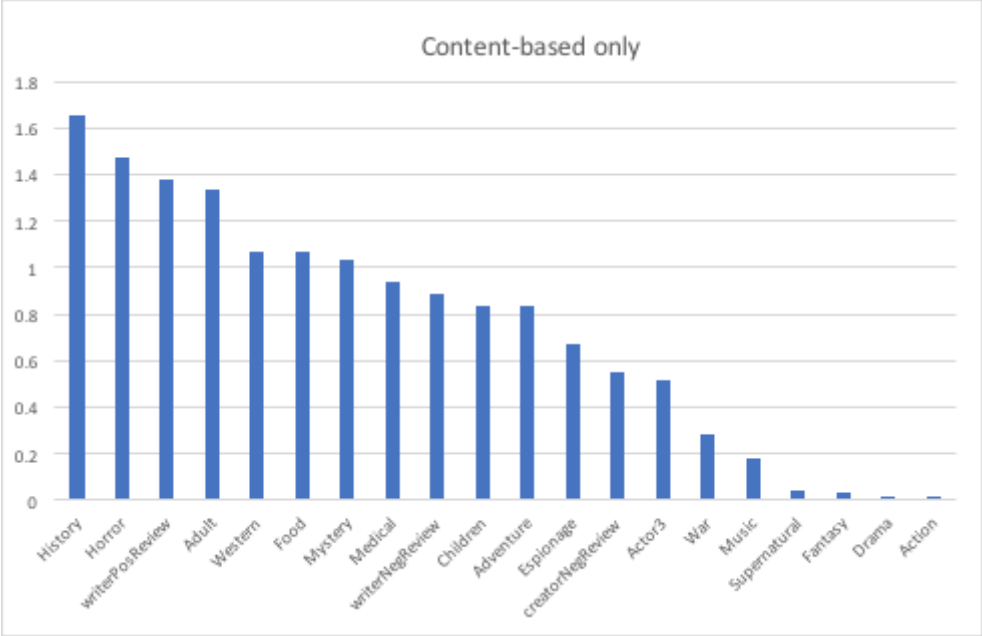

Figure 5. Optimized Feature Weighting from Content-based Only Attributes

In Figure 6, although added with episode related attributes, genre related attributes still top up and dominate the important features chart. While in Figure 7, after we added it with all the attributes, with a lot of channel-related attributes, it seems that channel related attributes become the most important ones, and they influence the similarity between TV shows.

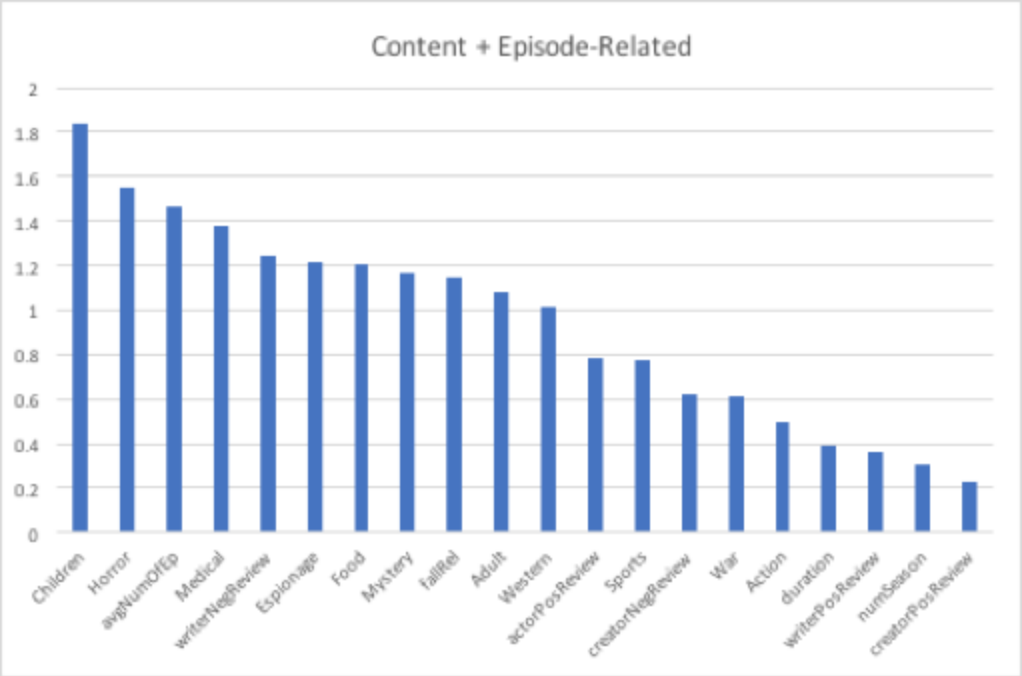

Figure 6. Optimized Feature Weighting from Content-based and Episode-related Attributes

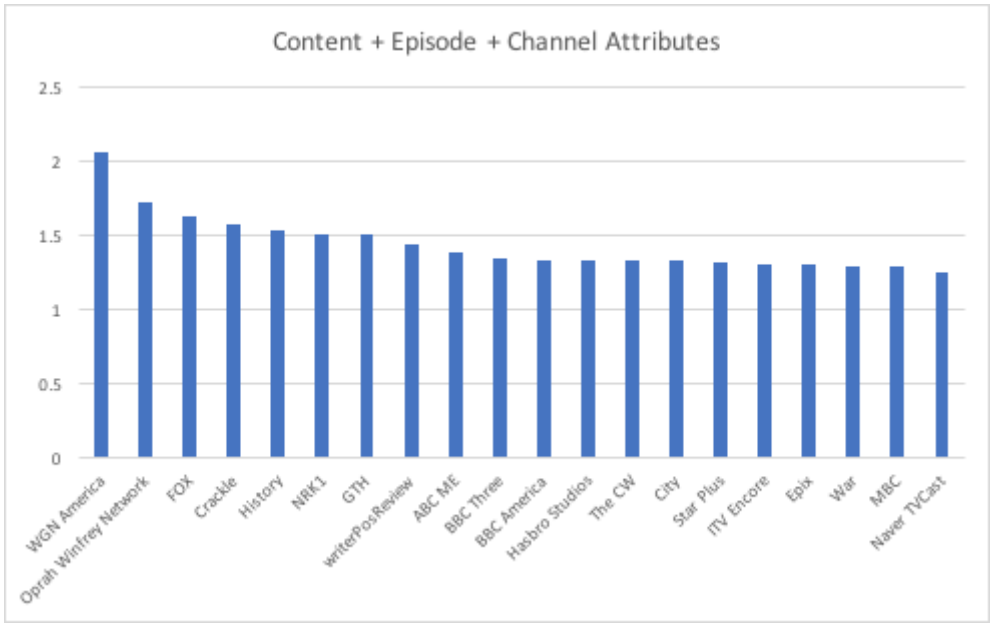

Figure 7. Optimized Feature Weighting from all the Attributes

Table 4 sorts out the top five important features in three different cases.

Data Source	A	A + B	A+B+C
f1 v1^2	GENRE_HISTORY 1.65	GENRE_CHILDREN 1.83	CH_WGN_AMERICA 2.05
f2 v2^2	GENRE_HORROR 1.47	GENRE_HORROR 1.55	CH_OPRAH_NETWORK 1.72
f3 v3^2	WRITER_POS_REVIEW 1.37	AVG_NUM_EPISODE 1.46	CH_FOX 1.62
f4 v4^2	GENRE_ADULT 1.33	GENRE_MEDICAL 1.38	CH_CRACKLE 1.56
f5 v5^2	GENRE_WESTERN 1.07	WRITER_NEG_REVIEW 1.13	GENRE_HISTORY 1.52

Table 4. The Top Five Important Attributes for Three different Cases

6.3 Compare Output against Hypothesis

For hypothesis 1: Content-based (TV series-based) data are the main factors to influence the accuracy of the prediction of ratings.

1. As the performance metrics indicate, this assumption is wrong. The Release-based data outperform the one with content-based data. The possible reasons are as followed: the number of the release-based attributes(175) is way more than the number of content-based attributes(39). If we could cleanse release-based data well and adequately, perhaps we will not get this conclusion.
2. Secondly, the performance metric is built upon MSE, and the optimization procedure is aiming at minimizing the MSE. As we see from the prediction result, we only predict well in the "GOOD" category. We do not predict well in "VERY GOOD" and "POOR" category. Perhaps aiming only on MSE is not a good choice.

For hypothesis 2: There will mainly be three kinds of data sources that correspond to different phases for a TV series, content-based data for production phase, release-based data for pre-release phase, and audience-based data for post-release phase. As the time go on and we get different types of data sources, we could predict the ratings more and more accurately.

1. as shown in Table 1, we could get minimized MSE if we use all the attributes and extract

the important concepts amongst all the attributes. However, in some cases, the MSE does not lower as we expected with more attributes.

6.4 Abnormal Case Explanation

As we find out that the predicted rating does not linearly track the actual measured rating in Figure 4, we go track the TV shows with the significant MSE. In Table 5, for a target TV show, we can see that his similar TV shows do not share the similar rating. The reason could be the most famous TV shows are outstanding. These successful TV shows are in some sense different than other mediocres. On the other hand, the reason why we predict the "GOOD" category well is that mediocre TV shows are a lot, and they normally have lots of similar TV shows.

Data Source	84: Anne	46: Lovesick	192: The Doctor Blake Mysteries
Rating	9.6	9.3	9.4
Predicted Rating	7.82	7.6	7.91
1st similar show	The last Tycoon 7.6	Casual 8.3	Murder in the First 7.7
2nd similar show	Pitch 7.8	Fleabag 7.4	Stan Lee's Lucky Man 8.1
3rd similar show	Halt and Catch Fire 8.1	Those Who Can't 8	100 Code 8.3

Table 5. Abnormal Cases with High Actual Rating

6.5 Discussion

Based on the previously presented results it becomes quite clear that kernel-based model requires a hypothesis that similar TV show should have similar rating. Otherwise, the prediction will be based upon the wrong thing. Moreover, how we define similar should be carefully investigated as well. In the last case, with all the data sources, channel-based attributes dominate the

important features and they determine the similarity between TV shows. It is worth investigating deeper to see why this come out.

VII. Conclusions and Recommendations

7.1 Summary and conclusions

This project mainly utilized kernel-based models to predict viewer's ratings of TV series based on the existing IMDb database and other secondary database. In particular, the classification model with ratings divided into three subgroups provides the best outcome and is recommended for prediction, although the outcome falls in a relatively wider range. Nevertheless, kernel-based using selected features, either by using weighting or PCA(SVD), provides improved results compared to equally-weighted with all available features. If given more time, we will try using more sophisticated models to run on the data, for example, neuro-network.

7.2 Recommendations for future studies

Looking back at this experiment and results but also the TV show domain as a whole it becomes clear that there is still more work to be done within the area of TV show prediction. This section aims to present an overview of the relevant future work that has been identified throughout the study.

The first and possibly the most obvious improvement that come to mind is the TV show attributes used in the study. As noted in previous sections, the current set of TV show attributes only seems to describe part of the general user rating of a TV show, making to believe that further improvements and research on the factors and relations within TV show data would be highly beneficial for the overall quality of TV show predictions. When looking at the variable importance of the experiment data identified by the kernel-based algorithm (Figure 3), the channel the TV show is broadcast on seems to be an important factor to account for when

predicting movie ratings, and might therefore be worth to develop and simplify further; especially considering that there seems to be no consensus on how to represent this type of information within previous studies.

Further on it might also be relevant to evaluate whether or not the prediction performance and results are generalizable over multiple datasets as well as over larger and smaller datasets in order to be able to draw more general conclusions, as the current experiment setup only incorporated a single though well established dataset. Likewise, it might also be of interest to evaluate whether or not the same is applicable to hit rate predictions, to further widen the scope of the study.

Finally, we will also analyze the TV series ratings based on different time spans to see how the importance of features changes over time.

VIII. Bibliography

1. Jehoshua Eliashberg, Sam K. Hui, and Z. John Zhang. “Assessing Box Office Performance Using Movie Scripts: A Kernel-Based Approach.” IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 26, NO. 11, NOVEMBER 2014

2. Hidenari IWAI, Yoshinori HIJIKATA, Kaori IKEDA, Shogo NISHIDA. “Sentence-based Plot Classification for Online Review Comments.” 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies(IAT)

3. Krushikanth R. Apala, Merin Jose, Supreme Motnam, C.-C. Chan, Kathy J. Liszka, and Federico de Gregorio. “Prediction of Movies Box Office Performance Using Social Media.” 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining

4. Gradient-free algorithm
http://adl.stanford.edu/aa222/lecture_notes_files/chapter6_gradfree.pdf

IX. Appendices

- Prediction program source code

a) SVD part

```
//package com.company;

/**
* Created by yangyaochia on 26/05/2017.
*/
import org.ejml.data.Matrix;
import org.ejml.simple.SimpleMatrix;
import org.ejml.simple.SimpleSVD;

import javax.sound.midi.SysexMessage;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.ArrayList;

public class SVD {

  private double[][] m;
  private double[][] u;
  private double[][] w;
  private double[][] v;
  private double[][] uw;

  public SVD(double[][] mat) {

    this.m = mat;
  }

  public void buildSVD() {
    SimpleMatrix A = new SimpleMatrix(m);
    //Matrix AA = new Matrix(m);

    /*for( int i = 0; i < A.numRows(); i++) {
      for( int j = 0 ; j < A.numCols() ; j++) {
        System.out.print(A.get(i,j) + " ");
      }
      System.out.println();
    }*/
    //System.out.println("----------------------------------");
    //SimpleMatrix matA = new SimpleMatrix(matrixData);
    @SuppressWarnings("unchecked")
    SimpleSVD svd = A.svd();

    SimpleMatrix U = (SimpleMatrix) svd.getU();
    SimpleMatrix W = (SimpleMatrix) svd.getW();
```

```
SimpleMatrix V = (SimpleMatrix) svd.geTV();

//System.out.println("U row = "+U.numRows()+" U col = "+U.numCols());
//System.out.println("W row = "+W.numRows()+" W col = "+W.numCols());
//System.out.println("V row = "+V.numRows()+" V col = "+V.numCols());

double sum = 0;
int indexW = ( W.numCols() < W.numRows() )? W.numCols():W.numRows();
//System.out.println("indexW = "+indexW);
for ( int i = 0 ; i < indexW ; i++ )
   sum += Math.pow(W.get(i,i), 2);
int accumuIndex = 0;
double accumuSum = 0;
for ( int i = 0 ; i < W.numCols() ; i++ ) {
   accumuSum += Math.pow(W.get(i,i), 2);
   if ( accumuSum/sum >= 0.98 ) {
      accumuIndex = i;
      break;
   }
}
//System.out.println("accum index = "+accumuIndex);

u = new double[U.numRows()][accumuIndex];
w = new double[accumuIndex][accumuIndex];
v = new double[accumuIndex][V.numCols()];
uw = new double[U.numRows()][accumuIndex];

//System.out.println("U row = "+u.length+" U col = "+u[0].length);
//System.out.println("W row = "+w.length+" W col = "+w[0].length);
//System.out.println("V row = "+v.length+" V col = "+v[0].length);
for( int i = 0; i < u.length; i++) {
   for( int j = 0 ; j < u[0].length ; j++) {
      u[i][j] = U.get(i,j);
   }
}
//System.out.println("---------------------------------");
for( int i = 0; i < w.length; i++) {
   for( int j = 0 ; j < w[0].length  ; j++){
      w[i][j] = W.get(i,j);
      //System.out.print(W.get(i,j) + " ");
   }

   //System.out.println();
}
//System.out.println("---------------------------------");
for( int i = 0; i < v.length; i++) {
   for( int j = 0 ; j <  v[0].length  ; j++) {
      v[i][j] = V.get(i,j);
      //System.out.print(V.get(i,j) + " ");
   }

   //System.out.println();
}
for (int i = 0; i < u.length; i++) { // aRow
   for (int j = 0; j < w.length; j++) { // bColumn
```

```
            for (int k = 0; k < u[i].length; k++) { // aColumn
                uw[i][j] += u[i][k] * w[k][j];
            }
        }
    }

}
public void printU()
{
    try{
        //All your IO Operations
        FileWriter fw = new FileWriter("U.csv");
        for( int i = 0; i < u.length; i++) {
            for( int j = 0 ; j <  u[i].length  ; j++) {
                //v[i][j] = V.get(i,j);
                //System.out.print( fwd(u[i][j],11,4) );
                fw.append(String.valueOf(u[i][j])+" ");

            }

            fw.append("\n");
        }
        fw.append("\n");
        fw.close();
    }
    catch(IOException ioe){
        //Handle exception here, most of the time you will just log it.
    }
}
public void printUW()
{
    try{
        //All your IO Operations
        FileWriter fw = new FileWriter("UW.csv");
        for( int i = 0; i < uw.length; i++) {
            for( int j = 0 ; j <  uw[i].length  ; j++) {
                //v[i][j] = V.get(i,j);
                //System.out.print( fwd(u[i][j],11,4) );
                fw.append(String.valueOf(uw[i][j])+" ");

            }

            fw.append("\n");
        }
        fw.append("\n");
        fw.close();
    }
    catch(IOException ioe){
        //Handle exception here, most of the time you will just log it.
    }
}
static String fwd(double x, int w, int d)
// converts a double to a string with given width and decimals.
{
    java.text.DecimalFormat df = new DecimalFormat();
```

```
        df.setMaximumFractionDigits(d);
        df.setMinimumFractionDigits(d);
        df.setGroupingUsed(false);
        String s = df.format(x);
        while (s.length() < w)
            s = " " + s;
        if (s.length() > w)
        {
            s = "";
            for (int i=0; i<w; i++)
                s = s + "-";
        }
        return s;
    }

    public double[][] getU()
    {
        return u;
    }
    public double[][] getUW()
    {
        return uw;
    }

}
```

b) Prediction part

```
/**
* Created by yangyaochia on 05/06/2017.
*/

import java.io.FileWriter;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.*;


public class Prediction {

    private double[][] m;
    private ArrayList<Double> rating;
    private NelderMead findFeatureWeighting = null;
    private int kFold;
    private double[] error;
    private int numTesting;
    private int numTraining;
    private double[][] matTesting;
    private double[][] matTraining;

    private ArrayList<Double> ratingTesting;
    private ArrayList<Double> ratingTraining;
```

```
private double[] predictedRating;

private double[][] distance;
private double[][] weightings;
private double[][] featureWeighting;

private double[][] finalDistance;
private double[][] finalWeightings;
private double[]   finalPredictedRating;

static private double THETA;
private static int count1=0;
private static int count2=0;
private static int count3=0;

//static private double[][] errorTesting;

public Prediction(double[][] mat, ArrayList<Double> rating, int kFold) {
   this.m = mat;
   this.rating = rating;
   this.kFold = kFold;
   this.error = new double[kFold];
   this.numTesting = mat.length/kFold;
   this.numTraining = (mat.length*(kFold-1)/kFold == 0)? numTesting:mat.length*(kFold-1)/kFold;

   this.predictedRating = new double[numTesting];
   this.distance = new double [numTesting][numTraining];

   this.weightings = new double [numTesting][numTraining];

   this.finalDistance = new double [mat.length][mat.length];
   this.finalWeightings = new double [mat.length][mat.length];
   this.finalPredictedRating = new double [mat.length];

   this.matTraining = new double[numTraining][mat[0].length];
   this.matTesting = new double[numTesting][mat[0].length];

   this.featureWeighting = new double[kFold+1][mat[0].length];

   this.ratingTesting = new ArrayList<Double>();
   for (int i = 0 ; i < numTesting ; i++)
      ratingTesting.add(0.0);

   this.ratingTraining = new ArrayList<Double>();
   for (int i = 0 ; i < numTraining ; i++)
      ratingTraining.add(0.0);

   //this.errorTesting = new double[kFold][20];
}

public void predict()
{

   for ( int i = 0 ; i < kFold ; i++ ) {
```

```
        buildMatTraining(i);
        buildMatTesting(i);

        buildRatingTraining(i);
        buildRatingTesting(i);

        //System.out.println("matTraining row = "+matTraining.length+" col = "+matTraining[0].length);
        //System.out.println("matTesting row = "+matTesting.length+" col = "+matTesting[0].length);
        //System.out.println("ratingTraining row = "+ratingTraining.size());
        //System.out.println("ratingTesting row = "+ratingTesting.size() );

        findFeatureWeighting = new NelderMead(matTraining, ratingTraining);

        findFeatureWeighting.descend();
        //findFeatureWeighting.printRating(i);
        //findFeatureWeighting.printPredictedRating(i);
        //findFeatureWeighting.printMat(i);
        //findFeatureWeighting.printFeatureWeighting(i);
        //findFeatureWeighting.printDistance(i);
        //findFeatureWeighting.printWeightings(i);
        //System.out.println("MSE = "+findFeatureWeighting.getMSE());
        double[] v = findFeatureWeighting.getFeatureWeighting();

        for (int j = 0 ; j < v.length ; j++ ) {
            featureWeighting[i][j] = v[j];
        }

        THETA = findFeatureWeighting.getTheta();
        error[i] = getError(v, i);

        //printRatTrain(i);
        //printRatTest(i);

    }
    for ( int j = 0 ; j < featureWeighting[0].length ; j++ )
    {
        for ( int i = 0 ; i < featureWeighting.length - 1 ; i++ ) {
            featureWeighting[kFold][j] += SQR(featureWeighting[i][j]);
        }
        featureWeighting[kFold][j] = Math.sqrt(featureWeighting[kFold][j]/kFold);
    }
    printFinalFeatureWeighting();
    printFinalPredictedRating();
    printActualRating();

  }

private void printFinalFeatureWeighting() {
  try{
    //All your IO Operations
    FileWriter fw = new FileWriter("FinalFeatureWeighting"+ count1 +".csv");
    for (int index = 0; index < featureWeighting[kFold].length; index++)
    {
        fw.append(String.valueOf(featureWeighting[kFold][index])+"\n");
```

```
      }
      fw.append("\n");
      fw.close();
    }
    catch(IOException ioe){
      //Handle exception here, most of the time you will just log it.
    }

    count1++;
}

private void buildMatTraining(int iTimes){
    for ( int i = 0 ; i < matTraining.length ; i++ ) {
      for ( int j = 0 ; j < matTraining[i].length ; j++ ) {
        matTraining[i][j] = m[(i + iTimes * numTraining) % m.length][j];
      }
    }

}
private void buildMatTesting(int iTimes){
    for ( int i = 0 ; i < matTesting.length ; i++ ) {
      for ( int j = 0 ; j < matTesting[i].length ; j++ ) {
        matTesting[i][j] = m[(i + (iTimes+1) * numTraining)% m.length][j];
      }
    }
}
private void buildRatingTraining(int iTimes){

    for ( int i = 0 ; i < ratingTraining.size() ; i++ ) {
      ratingTraining.set(i, rating.get((i + iTimes * numTraining) % m.length));
    }
}
private void buildRatingTesting(int iTimes){

    for ( int i = 0 ; i < ratingTesting.size() ; i++ ) {
      ratingTesting.set(i, rating.get( ((i + (iTimes+1) * numTraining) % m.length)) );
    }

}
private double getError(double v[], int iTimes) {
    double error = 0;

    for ( int i = 0 ; i < matTesting.length ; i++ ) {
      error += SQR(ratingTesting.get(i) - predictingRating(i, v, iTimes));
      //System.out.println("i = "+i+": Actual raing = "+ratingTesting.get(i)+" , Predicted raing = "+predictedRating[i]);
    }
    error /= matTesting.length;
    return error;
}

double predictingRating(int indexPredicted, double v[], int iTimes)
{
    double tempDistance = 0;

    for ( int i = 0 ; i < matTraining.length ; i++ ) {
```

```
      //if ( i == indexPredicted )
      //    continue;
      for ( int j = 0 ; j < matTraining[0].length ; j++ ) {
         tempDistance += SQR(v[j] * (matTesting[indexPredicted][j] - matTraining[i][j]) );
      }
      distance[indexPredicted][i] = Math.sqrt(tempDistance);
      tempDistance = 0;
   }

   for ( int i = 0 ; i < matTraining.length ; i++ ) {
      weightings[indexPredicted][i] = Math.exp(-1 * THETA * distance[indexPredicted][i]);
   }
   double sum = 0;
   for ( int i = 0 ; i < matTraining.length ; i++ ) {
      //if ( i != indexPredicted ) {
      sum += weightings[indexPredicted][i];
      //}
   }
   for ( int i = 0 ; i < matTraining.length ; i++ ) {
      //if ( i != indexPredicted ) {
      weightings[indexPredicted][i] /= sum;
      //}
   }

   double predictedResult = 0;
   for ( int i = 0 ; i < matTraining.length ; i++ ) {
      //if ( i != indexPredicted ) {
      predictedResult += weightings[indexPredicted][i] * ratingTraining.get(i);
      //System.out.println(fwd(weightings[indexPredicted][i], 8, 4)+" "+fwd(ratingTraining.get(i),8,2));
      //}
   }
   predictedRating[indexPredicted] = predictedResult;
   finalPredictedRating[((indexPredicted + (iTimes+1) * numTraining) % m.length)] = predictedResult;
   return predictedResult;
}

   double SQR(double x)
   {
      return x*x;
   }

   public double[] getFinalPredictedRating(){
      return finalPredictedRating;
   }

   void printFinalPredictedRating(){
      try{
         //All your IO Operations
         FileWriter fw = new FileWriter("Final_Predicted_Rating"+count2+".csv");
         for (int index = 0; index < finalPredictedRating.length; index++)
         {
            fw.append(String.valueOf(finalPredictedRating[index])+"\n");
         }
         fw.append("\n");
         fw.close();
```

```
    }
    catch(IOException ioe){
        //Handle exception here, most of the time you will just log it.
    }

    count2++;
}
void printActualRating() {
    try{
        //All your IO Operations
        FileWriter fw = new FileWriter("Actual_Rating"+count3+".csv");
        for (int index = 0; index < rating.size(); index++)
        {
            fw.append(String.valueOf(rating.get(index))+"\n");
        }
        fw.append("\n");
        fw.close();
    }
    catch(IOException ioe){
        //Handle exception here, most of the time you will just log it.
    }
    count3++;
}

double getMeanSquaredError()
{
    double result = 0;
    for (int i = 0 ; i < error.length ; i++ )
    {
        result += error[i];
        System.out.print(error[i]+" ");
    }
    System.out.println();
    result /= error.length;
    return result;
}
public void printRatTrain(int i){
    try{
        //All your IO Operations
        FileWriter fw = new FileWriter("rating_training"+i+".csv");
        for (int index = 0; index < ratingTraining.size(); index++)
        {
            fw.append(String.valueOf(ratingTraining.get(index))+"\n");
        }
        fw.append("\n");
        fw.close();
    }
    catch(IOException ioe){
        //Handle exception here, most of the time you will just log it.
    }
}
public void printRatTest(int i){
    try{
        //All your IO Operations
        FileWriter fw = new FileWriter("rating_testing"+i+".csv");
```

```
            for (int index = 0; index < ratingTesting.size(); index++)
            {
                fw.append(String.valueOf(ratingTesting.get(index))+"\n");
            }
            fw.append("\n");
            fw.close();
        }
        catch(IOException ioe){
            //Handle exception here, most of the time you will just log it.
        }
    }

    String fwd(double x, int w, int d)
    // converts a double to a string with given width and decimals.
    {
        java.text.DecimalFormat df = new DecimalFormat();
        df.setMaximumFractionDigits(d);
        df.setMinimumFractionDigits(d);
        df.setGroupingUsed(false);
        String s = df.format(x);
        while (s.length() < w)
            s = " " + s;
        if (s.length() > w) {
            s = "";
            for (int i=0; i<w; i++)
                s = s + "-";
        }
        return s;
    }
}
```