# Using Naïve Bayes to Predict Movie Ratings

Ryan Beck, Nick Moore, Torin Stetina, Andrew Wildman
AMATH 582
University of Washington, Seattle

March 16, 2017

**Abstract**

Understanding which movies will be box-office hits is a constant problem in the movie industry. In this paper, we propose a Naïve Bayes based algorithm to predict IMDb ratings based on the plot descriptions of the movies.

## Introduction

If one were to be able to predict whether or not a movie would be a blockbuster based solely on the plot description, they could easily make many hit movies. This paper proposes an algorithm to do half of that: predict ratings based upon plot descriptions. In essence, this problem could be framed as either categorization or regression. That is to say, ratings can either be seen as continuous – a scale – or as discrete – good and bad. In this paper, we choose to model ratings as a categorical variable, since models based upon similar assumptions have produced good results.

Filtering spam emails is a similar problem. Commonly, spam filters attempt to extract information from emails and predict if the message is useful or not. Many machine learning techniques have been applied to this, and one of the techniques producing the best separation of spam and non-spam messages is Naïve Bayes. This technique is effective enough that it has even been implemented in email software packages [1]. In this report, we adapt the methodology used for spam detection in order to predict ratings based off the plot descriptions of movies.

First, we will introduce the theory behind Naïve Bayes classification and remind readers of the structure of a multinomial distribution. Next, we will go through the details of our implementation and the reasons behind some of our choices. Then we will present the results of applying our algorithm to a variety of test cases. Finally, we will discuss the strengths and limitations of our ratings prediction algorithm, and we will provide the source code for our algorithm.

## Theoretical Background

Naïve Bayes is a probabilistic model for classifying groups of data, the likes of which also include decision-tree learning and k nearest neighbors classification. Since this is a supervised learning model, the training set for the data is labeled by the user before the Naïve Bayes algorithm begins. This algorithm is fast and was a suitable choice for "bag of words" type classifications where there are many labels to be used in the dataset. The consequences of using this algorithm will be discussed in Computational Results.

Naïve Bayes is based upon Bayes' theorem, which allows us to describe a conditional probability in terms of a prior, it's likelihood, and the evidence. More specifically, we can take a data set $\boldsymbol{X}$, whose elements $\boldsymbol{x}$ represent single data measurements with N variables $x_k$. Let $c_i$ denote a classification to which $\boldsymbol{x}$ can belong. The probability of $\boldsymbol{x}$ belonging to the class $c_i$ can then be defined, given the actual measurement $\boldsymbol{x}$, as:

$$p(c_i|\boldsymbol{x}) = \frac{p(c_i)p(\boldsymbol{x}|c_i)}{p(\boldsymbol{x})} \tag{1}$$

Now we can apply the "Naive" assumption which allows us to classify the variables. If we assume all elements of $x_k$ are independent, then the probability of measuring $\boldsymbol{x}$ in the class $c_i$ is given by the following equation

$$p(\boldsymbol{x}|c_i) = \prod_{k=1}^{N} p(x_k|c_i) \tag{2}$$

Which lets us reduce everything down (noting that $p(\boldsymbol{x})$ is known when we have $\boldsymbol{\mathcal{X}}$) to:

$$p(c_i|\boldsymbol{x}) = \frac{p(c_i)}{p(\boldsymbol{x})} \prod_{k=1}^{N} p(x_k|c_i) \tag{3}$$

The term, $p(c_i)$ is calculated from the dataset given. In order to define $p(x_k|c_i)$ in practice, a probability distribution is assumed for each variable, and used for calculations of the probabilites. For this work, a multinomial distribution [2] was assumed for each variable, giving

$$p(x_k|c_i) = \frac{\left(\sum_i x_i\right)!}{\prod_i x_i!} \prod_i (\lambda_{ki})^{x_i} \tag{4}$$

where $x_i$ is the frequency count of a variable in the dataset $\boldsymbol{x}$ and $\lambda_{ki}$ is the $i^{th}$ parameter of the $k^{th}$ observation in the set $\boldsymbol{\mathcal{X}}$.

## Algorithm Development and Implementation

The goal of our algorithm is to produce a movie's plot representation in a frequent-words space (hereafter referred to as the word space), and then use Naïve Bayes in order to predict their ratings. Note that, since Naïve Bayes is a classification method and not a regression method, the predicted rating is categorical, not continuous.

Prior to analyzing the data, the data must be gathered and cleaned. All data gathered for this project was generated from files downloaded from the IMDb plain text data files FTP servers. In particular, the files downloaded were "ratings.list" and "plots.list." The ratings.list file was parsed and the average rating and movie title for all movies with greater than 2000 reviews were extracted. This list was then cross-checked against the plots.list file, and the name of the movie, the rating and the plot were extracted if the movie existed in both the ratings file and the plots file. 18,481 movies remained after this process.

The next step to producing the word space is to generate a dictionary of frequent words. Counting the occurrences of each word would promote sparsity in our word space and would not improve predictive power, since "possibly" and "possible" have essentially the same meaning but would be mapped to two different variables. In order to reduce the size of the dictionary and collapse some words with similar meaning into one token, we stemmed each word prior to addition to the dictionary. Stemming is a process by which the inflection of words is reduced to the stem or root form. We employed the Natural Language Toolkit implementation of the Porter stemmer [3]. Since, for our purposes, the stems did not need to be english words, the Porter stemming algorithm seemed to be a good fit for our data. Each stem was collected in a file along with the frequency of its occurrence across all movie plots, finalizing our dictionary with 64,444 entries.

In order to make the plot representations in the word space computationally tractable, the effective dictionary used was truncated. Most entries in the finalized dictionary wouldn't produce a large separation of clusters in word space, since they only occurred a few times over our large set of movies. For example, the last 27,765 entries only occurred once in 18,481 movies, so they can't be used to separate differently rated movies well. Furthermore, we expect the representation of all movies to be essentially the same for common words such as "the," "to," or "a," so we excluded the first 49 most common words. The truncated dictionary consisted of taking the 50th to 1001st most common stems (952 total) as the word space.

The representations of the plots in this word space were generated by maintaining order of frequency in all movies overall, and then counting the occurrence of each stem within each plot. This process created a data matrix with each row corresponding to a movie, the first column corresponding to the average rating of that movie, and the remaining columns consisting of the representation of the movie in the word space. It is worthwhile to note that all the processing up to this point was performed in python (See Appendix B), but the naïve bayes analysis was performed in MATLAB. The data matrix produced by this final python script was in the correct format for MATLAB to read using the tblread(·) function (See Appendix A).

After the data matrix was read into MATLAB and the ratings column separated from the others, the built-in naïve bayes classification was performed on the data, using a variety of different categories, discussed in Computational Results. The naïve bayes classifiers were trained with a set of 80% of each category and tested against the remaining 20%. Since the parameters of each observation are discrete variables, the standard (gaussian) probability distribution would be improper in this case. Instead, we used a multinomial distribution in order to model each parameter. We also did not take a singular value decomposition of the data matrix, since it would have mapped the discrete variables into a continuous space. The naïve bayes classifiers were cross-validated enough times ($N \geq 200$) to stabilize the average fraction that each category was classified correctly.

## Computational Results

Nave Bayes classification with cross validation was used to test a set of movie plot descriptions against the set of ratings for those movies. For the first trial, all movies in the data set were collected into a

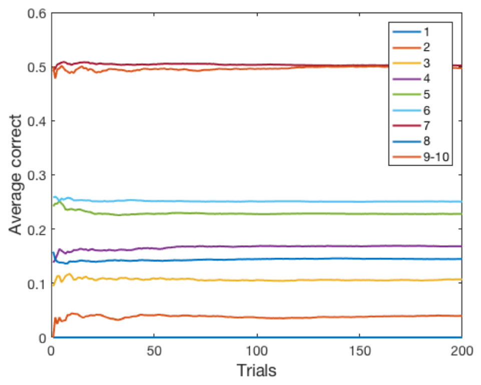

**Figure 1:** Plot of average correct classifications vs. number of trials used over 200 cross validation tests. The integer numbers in the legend correspond to the rating bin.

category corresponding to the movie's rating (a rating bin), rounded down. For example, movies with a rating in the range [2, 3) were placed in the category labeled 2. The one exception to this method was the category labeled 9-10, since it spanned [9, 10]. These categorized movies were then used to train the predictive model, and cross validated 200 times. For this set of data, 200 cross validation runs was enough to stabilize the average fraction correctly predicted in each category (Figure 1). The predictive power of this model was quite poor. Since there are 9 bins in this organization of the data, the probability of being correct after placing any movie into a random bin is about 11%, so this is the absolute baseline we must compare our data against. As can be seen in Figure 1, our model predicted three of these nine categories less effectively than random chance. Most of the others have only marginal improvement over random chance, but two of the bins, 7 and 9-10, are significantly more accurate than the others, with both achieving 50% correctly identified. In order to gain insight into the unequal ability of our model to predict ratings, the misclassification of the poorly performing ratings must be inspected. Figure 2 shows a plot of the proportion of movies in each category. For example, for the movies with an actual rating of 7 (the burnt-orange colored bars), the proportion of movies classified as a 5 is given by the orange bar on the category '5.' Note that the distribution of proportions tends to be the same across all runs, with the exception of the 9-10 bin. Since we took all movies in this trial, the numbers of movies in each rating bin were different which caused the movies with a low number of ratings to have a low accuracy in our model. As can be seen by comparison to Figure 3, the distribution of classification proportions roughly follows the distribution of ratings.

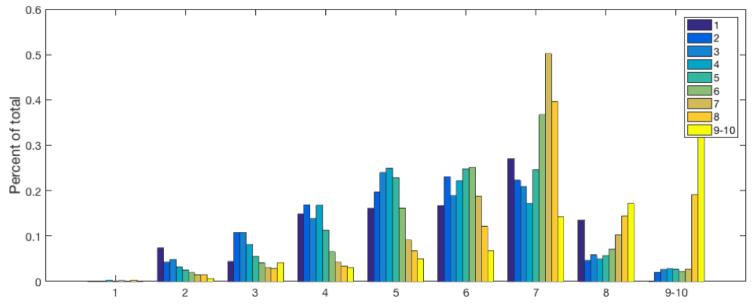

**Figure 2:** Histogram of fraction correct for the classification from the Naïve Bayes algorithm in trial 1.

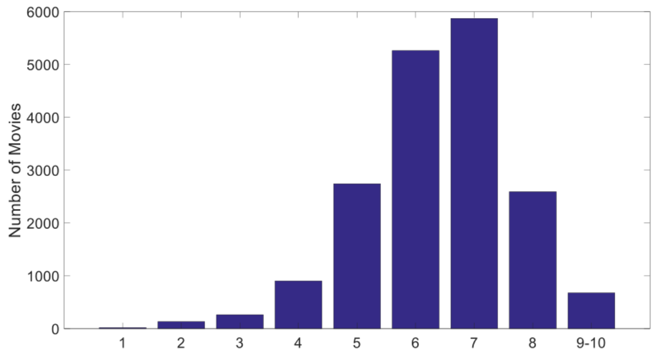

**Figure 3:** Histogram of rating occurrence for all movies

The effect of this bias can be seen when comparing trials 1 and 2. Trial 2, the results shown in figure 4, removed the rating 1 bin and used an equal number of movies in all bins. In figure 2 the percentage correctly predicted increases with rating number, until bin 7 before dropping off. In contrast to this the percentage correctly predicted in Figure 4 is relatively equal for all bins except for 8 and 9-10, which increased the percentage correctly predicted from 15% and 50% to 48% and 63%, respectively. The difference between the results of trails 1 and 2 displays the importance of the number of movies used in each bin. Despite the improved results for 8 and 9-10, the other bins fared worse with percentages correctly identified ranging from 18% to 24%. With 8 different bins randomly assigning the ratings would have resulted in 12.5% predicted correctly, so our model is only slightly better than random selection for all bins. The number of bins was then reduced, in order to try to

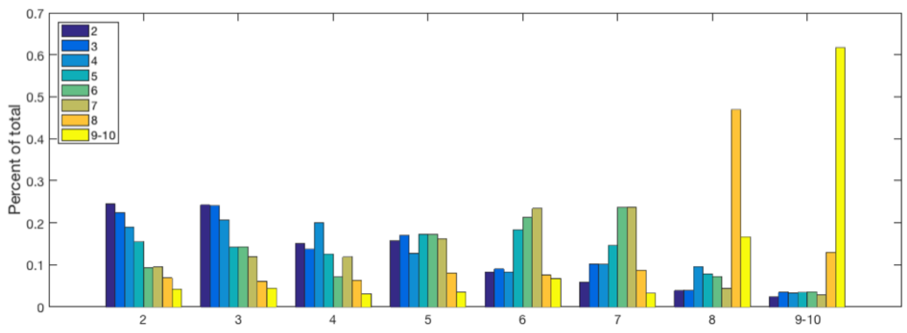

**Figure 4:** Plot of average correct classifications vs. number of trials used over 500 cross validation tests. The integer numbers represent the bin the a movie's rating belongs to. For example, a movie with a rating of 7.2 or 7.8 would go into the 7 bin. Overall, this data tells us the the cross validation has stable results after 200 tests.

improve the predictive power of the model. Trial 3 separated the movies into three different rating bins, low (1-3), medium (4-6), and high (7-10). Figure 5 shows the histogram plot of the results. The

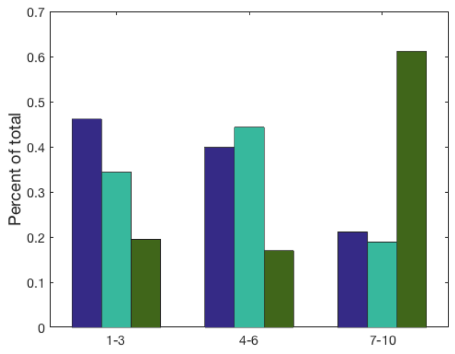

**Figure 5:** Histogram of percentage correct classifications for trial 3. Movies were binned into low (1-3), medium (4-6), and high (7-10) ratings.

more coarsely binned results are promising, with the percentage correctly identified of 46%, 44%, and 62% for low, medium, and high, compared to a random selection probability of 33%. Low and medium

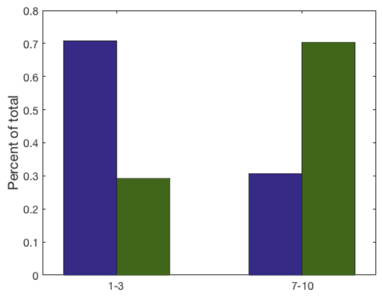

**Figure 6:** Histogram of percentage correct. Binned into low (1-3) and high (7-10) rating.

bins are roughly twice as accurate as they were in trial 2, and the high bin is relatively unchanged. Based on the results from trials 2 and 3 the high ranked movies have enough in class variance to be correctly identified even when in separate bins, but the low and medium ranked movies have little in class variance given the poor results from trials 1 and 2. The final trial consisted of only the high and low bins. Figure 6 shows the histogram plot of the results from trial 4. For both low and high bins the percentage correctly predicted is 70%, compared to an average probability of 50%. Another improvement over trial 3, but only slightly for the high rated movies, again supporting that the poor prediction is due to low and medium ranked movies.

## Conclusion

We have proposed an algorithm to predict movie ratings from plot descriptions. In general, our model improves with decreasing number of categories of ratings, and our algorithm is highly influenced by initial data distribution among the categories. We have improved prediction over random chance in all cases except the case in which the initial data distribution is non-uniform.

Our model only produces mild improvements over random selection due to a few fundamental problems. First is our choice of algorithm. While Naïve Bayes has been shown to be useful in filtering span, its applicability to text analysis problems with more than two categories may be limited. In particular, the assumption that frequency of any given word is independent of another is clearly incorrect, e.g. if “Family” occurs in a plot, then words such as “Father” or “Mother” would have a higher likelihood as well. Second is the fundamental sparsity of our data. Email messages, especially spam messages, tend to have less variety in topics than movie plots, so the word frequency space they occupy would be more densely populated.

In order to improve upon our algorithm, several approaches could be taken. First, one could take the SVD of our word frequency data, generating word frequency modes. This would help increase the statistical independence between elements, since words which commonly occur together would be expected to occur in the same mode. While this would fundamentally alter the structure of our data (mapping it from a discrete space to a continuous one), we could adapt our Naïve Bayes model fairly simply by modeling the projection of the movies onto the word-frequency modes as gaussian parameters. This may be a sub-optimal continuous distribution to model the projections, but it would be a start. Secondly, we could do regression instead of categorization. Since ratings can be treated as a continuous variable, we could use one of many regression techniques (such as partial least squares regression) to relate ratings and the word-frequency space. This would likely be most effective through the projection onto the word-frequency modes, since the types of variables used to regress would be the same.

## References

[1] Minoru Sasaki and Hiroyuki Shinnou. Spam detection using text clustering. In *Cyberworlds, 2005. International Conference on*, pages 4–pp. IEEE, 2005.

[2] Jason D Rennie, Lawrence Shih, Jaime Teevan, David R Karger, et al. Tackling the poor assumptions of naive bayes text classifiers. *ICML*, 3:616–623, 2003.

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

## Appendix A

To read the information from the text files given by the IMDb python scripts were constructed. The following extracts the ratings and pairs them with the plot summaries takes in files formatted from extract_ratings.py, and gives a file collecting the plots, names, and ratings in a tab delimited environment. Note that the delimiter writes on every line that does not have plot.

```
for rline in ratin:
t_split = re.split('::', rline.strip('\n'))
print(t_split)
names[t_split[1]] = t_split[0]

for line in plotin:
if 'MV:' in line:
pname = re.split('MV: ', line)[1].strip('\n')
if pname in names:
print(pname)
fout.write(pname + '::' + names[pname] + '::')
while '---------------------------------------------------------------------------------' not in l
if 'PL:' in line:
fout.write(re.split('PL: ', line)[1].strip('\n') + ' ')
line = plotin.next()
continue
fout.write('::')
line = plotin.next()
fout.write('\n')
```

The extract_ratings.py follows which takes in a minimum for the number of ratings, and maximum for the number of ratings, and the file from the IMDb.

```
filein = open(sys.argv[3],'r')
fileout = open('extracted_ratings.txt', 'w')

for line in filein:
linelist = re.split(' +',line)
name = linelist[4]
for part in linelist[5:]:
name += ' ' +  part
if float(linelist[2]) >= 1000:
if float(sys.argv[1]) <= float(linelist[3]) <= float(sys.argv[2]):
fileout.write('%s::%s' % (linelist[3], name))
```

To read information from the textfiles generated from the python code was the built in MATLAB function tblread() which outputs three values, the data from the file, the column headers from the tables in the file, and the left-most column of the file. The text files generated was deliminated with tabs.

```
[data, stems, movie] = tblread('pplots_1000.txt','\t');
```

## Appendix B

Code from MATLAB follows:

```
%%
clear; clc;

%%
%Import data
[data, stems, movie] = tblread('pplots_1000.txt','\t');

%%
%Extract ratings
ratings = data(:,1);
data2 = data(:,2:end);

ratings = floor(ratings);

%%
sind = zeros(1,9);
eind = zeros(1,9);
len = zeros(1,9);
for i = 1:9
temp = find(ratings==i);
sind(i) = temp(1);
eind(i) = temp(end);
len(i) = numel(temp);
end

tlen = floor(0.8*len);
clen = len - tlen;

trat = [];
crat = [];
for i = 1:9
trat = [trat; i*ones(tlen(i),1)];
crat = [crat; i*ones(clen(i),1)];
end
%%
nloop = 200;
correct = zeros(9,nloop);
freq = zeros(9);
tic;
for count = 1:nloop
train = [];
test = [];
for i = 1:9
perm = randperm(len(i));
train = [train; data2(perm(1:tlen(i))+sind(i),:)];
```

```
test = [test; data2(perm(tlen(i)+1:end)+sind(i),:)];
end

nb = fitcnb(train, trat, 'DistributionNames', 'mn');

pre = nb.predict(test);

correct(1, count) = sum(pre(1:clen(1)) == crat(1:clen(1)))/clen(1);
for i = 2:9
correct(i, count) = sum(pre(sum(clen(1:i-1)):sum(clen(1:i))) ==...
crat(sum(clen(1:i-1)):sum(clen(1:i))))/clen(i);
end

for i = 1:9
if i == 1
sind_t = 1;
else
sind_t = sum(clen(1:i-1));
end
eind_t = sum(clen(1:i));
for j = 1:9
freq(i,j) = freq(i,j) + numel(find(pre(sind_t:eind_t) == j));
end
end


if mod(count,10) == 0
toc;
count
tic;
end
end
toc;

%%
corr_avg=zeros(9,nloop);
for i = 1:nloop
for j =1:9
corr_avg(j,i) = sum(correct(j,1:i))/numel(correct(j,1:i));
end
end

pfreq = zeros(size(freq));
for i = 1:9
pfreq(i,:) = freq(i,:)/(nloop*clen(i));
end
```

```
%%
%NEW STRATEGY STARTS HERE

ratings = data(:,1);
data2 = data(:,2:end);

ratings = floor(ratings);
a1 = find(and(1 <= ratings, ratings <= 3));
a2 = find(and(3 < ratings, ratings <= 6));
a3 = find(and(6 < ratings, ratings <=10));
sind = [a1(1)-1, a2(1)-1, a3(1)-1];

len = [length(a1), length(a2), length(a3)];
tlen = floor(0.8*len);
clen = len - tlen;

ratings = [ones(1,length(a1)), 2*ones(1,length(a2)), 3*ones(1,length(a3))];
trat = [ones(1,tlen(1)), 2*ones(1,tlen(2)), 3*ones(1,tlen(3))];
crat = [ones(1,clen(1)), 2*ones(1,clen(2)), 3*ones(1,clen(3))];
%%
nloop = 200;
correct = zeros(3,nloop);
freq = zeros(3);
tic;
for count = 1:nloop
train = [];
test = [];
for i = 1:3
perm = randperm(len(i));
train = [train; data2(perm(1:tlen(i))+sind(i),:)];
test = [test; data2(perm(tlen(i)+1:end)+sind(i),:)];
end

nb = fitcnb(train, trat, 'DistributionNames', 'mn');

pre = nb.predict(test);

correct(1, count) = sum(pre(1:clen(1))' == crat(1:clen(1)))/clen(1);
for i = 2:3
correct(i, count) = sum(pre(sum(clen(1:i-1)):sum(clen(1:i)))' ==...
crat(sum(clen(1:i-1)):sum(clen(1:i))))/clen(i);
end

for i = 1:3
```

```
if i == 1
sind_t = 1;
else
sind_t = sum(clen(1:i-1));
end
eind_t = sum(clen(1:i));
for j = 1:3
freq(i,j) = freq(i,j) + numel(find(pre(sind_t:eind_t) == j));
end
end


if mod(count,10) == 0
toc;
count
tic;
end
end
toc;

%%
corr_avg=zeros(3,nloop);
for i = 1:nloop
for j =1:3
corr_avg(j,i) = sum(correct(j,1:i))/i;
end
end








%%
%TRIAL 3

ratings = data(:,1);
data2 = data(:,2:end);

ratings = floor(ratings);
a1 = find(and(1 <= ratings, ratings <= 3));
a2 = find(and(3 < ratings, ratings <= 6));
a3 = find(and(6 < ratings, ratings <=10));
sind = [a1(1)-1, a2(1)-1, a3(1)-1];

len = [length(a1)];
tlen = floor(0.8*len);
```

```
clen = len - tlen;

ratings = [ones(1,len), 2*ones(1,len), 3*ones(1,len)];
trat = [ones(1,tlen), 2*ones(1,tlen), 3*ones(1,tlen)];
crat = [ones(1,clen), 2*ones(1,clen), 3*ones(1,clen)];


%%
nloop = 500;
correct = zeros(3,nloop);
freq = zeros(3);
tic;
for count = 1:nloop
train = [];
test = [];
for i = 1:3
perm = randperm(len(1));
train = [train; data2(perm(1:tlen(1))+sind(i),:)];
test = [test; data2(perm(tlen(1)+1:end)+sind(i),:)];
end

nb = fitcnb(train, trat, 'DistributionNames', 'mn');

pre = nb.predict(test);

correct(1, count) = sum(pre(1:clen)' == crat(1:clen))/clen;
for i = 2:3
correct(i, count) = sum(pre((i-1)*clen:i*clen)' ==...
crat((i-1)*clen:i*clen))/clen;
end

for i = 1:3
if i == 1
sind_t = 1;
else
sind_t = (i-1)*clen;
end
eind_t = i*clen;
for j = 1:3
freq(i,j) = freq(i,j) + numel(find(pre(sind_t:eind_t) == j));
end
end


if mod(count,10) == 0
toc;
count
tic;
end
```

```
end
toc;


%%
corr_avg=zeros(3,nloop);
for i = 1:nloop
for j =1:3
corr_avg(j,i) = sum(correct(j,1:i))/i;
end
end

pfreq = freq/(nloop*clen);









%%
%TRIAL 4

ratings = data(:,1);
data2 = data(:,2:end);

ratings = floor(ratings);
a1 = find(and(1 <= ratings, ratings <= 3));
a3 = find(and(6 < ratings, ratings <=10));
sind = [a1(1)-1, a3(1)-1];

len = [length(a1)];
tlen = floor(0.8*len);
clen = len - tlen;

ratings = [ones(1,len), 2*ones(1,len)];
trat = [ones(1,tlen), 2*ones(1,tlen)];
crat = [ones(1,clen), 2*ones(1,clen)];

%%
nloop = 2000;
correct = zeros(2,nloop);
freq = zeros(2);
tic;
for count = 1:nloop
train = [];
```

```
test = [];
for i = 1:2
perm = randperm(len(1));
train = [train; data2(perm(1:tlen(1))+sind(i),:)];
test = [test; data2(perm(tlen(1)+1:end)+sind(i),:)];
end

nb = fitcnb(train, trat, 'DistributionNames', 'mn');

pre = nb.predict(test);

correct(1, count) = sum(pre(1:clen)' == crat(1:clen))/clen;
for i = 2:2
correct(i, count) = sum(pre((i-1)*clen:i*clen)' ==...
crat((i-1)*clen:i*clen))/clen;
end

for i = 1:2
if i == 1
sind_t = 1;
else
sind_t = (i-1)*clen;
end
eind_t = i*clen;
for j = 1:2
freq(i,j) = freq(i,j) + numel(find(pre(sind_t:eind_t) == j));
end
end


if mod(count,100) == 0
toc;
count
tic;
end
end
toc;

%%

corr_avg=zeros(2,nloop);
for i = 1:nloop
for j =1:2
corr_avg(j,i) = sum(correct(j,1:i))/i;
end
end

pfreq = freq/(nloop*clen);
```

```
%%
%TRIAL 5

ratings = data(:,1);
data2 = data(:,2:end);

ratings = floor(ratings);
a1 = find(and(4 <= ratings, ratings <= 6));
a3 = find(and(6 < ratings, ratings <=10));
sind = [a1(1)-1, a3(1)-1];

len = [length(a1)];
tlen = floor(0.8*len);
clen = len - tlen;

ratings = [ones(1,len), 2*ones(1,len)];
trat = [ones(1,tlen), 2*ones(1,tlen)];
crat = [ones(1,clen), 2*ones(1,clen)];

%%
nloop = 200;
correct = zeros(2,nloop);
freq = zeros(2);
tic;
for count = 1:nloop
train = [];
test = [];
for i = 1:2
perm = randperm(len(1));
train = [train; data2(perm(1:tlen(1))+sind(i),:)];
test = [test; data2(perm(tlen(1)+1:end)+sind(i),:)];
end

nb = fitcnb(train, trat, 'DistributionNames', 'mn');

pre = nb.predict(test);

correct(1, count) = sum(pre(1:clen)' == crat(1:clen))/clen;
for i = 2:2
correct(i, count) = sum(pre((i-1)*clen:i*clen)' ==...
crat((i-1)*clen:i*clen))/clen;
end
```

```
for i = 1:2
if i == 1
sind_t = 1;
else
sind_t = (i-1)*clen;
end
eind_t = i*clen;
for j = 1:2
freq(i,j) = freq(i,j) + numel(find(pre(sind_t:eind_t) == j));
end
end


if mod(count,10) == 0
toc;
count
tic;
end
end
toc;

%%

corr_avg=zeros(2,nloop);
for i = 1:nloop
for j =1:2
corr_avg(j,i) = sum(correct(j,1:i))/i;
end
end
```

```
%%
%TRIAL 5

ratings = data(:,1);
data2 = data(:,2:end);
```

```
ratings = floor(ratings);
a1 = find(and(1 <= ratings, ratings <= 3));
a3 = find(and(3 < ratings, ratings <=6));
sind = [a1(1)-1, a3(1)-1];

len = [length(a1)];
tlen = floor(0.8*len);
clen = len - tlen;

ratings = [ones(1,len), 2*ones(1,len)];
trat = [ones(1,tlen), 2*ones(1,tlen)];
crat = [ones(1,clen), 2*ones(1,clen)];

%%
nloop = 2000;
correct = zeros(2,nloop);
freq = zeros(2);
tic;
for count = 1:nloop
train = [];
test = [];
for i = 1:2
perm = randperm(len(1));
train = [train; data2(perm(1:tlen(1))+sind(i),:)];
test = [test; data2(perm(tlen(1)+1:end)+sind(i),:)];
end

nb = fitcnb(train, trat, 'DistributionNames', 'mn');

pre = nb.predict(test);

correct(1, count) = sum(pre(1:clen)' == crat(1:clen))/clen;
for i = 2:2
correct(i, count) = sum(pre((i-1)*clen:i*clen)' ==...
crat((i-1)*clen:i*clen))/clen;
end

for i = 1:2
if i == 1
sind_t = 1;
else
sind_t = (i-1)*clen;
end
eind_t = i*clen;
for j = 1:2
freq(i,j) = freq(i,j) + numel(find(pre(sind_t:eind_t) == j));
end
end
```

```
if mod(count,10) == 0
toc;
count
tic;
end
end
toc;

%%

corr_avg=zeros(2,nloop);
for i = 1:nloop
for j =1:2
corr_avg(j,i) = sum(correct(j,1:i))/i;
end
end


%%
%Plotting section
xticks = {'2','3','4','5','6','7','8','9-10'}

axsz = 14;
lbsz = 18;

figure(1); clf;
plot(corr_avg','LineWidth',2);
set(gca,'fontsize',axsz);
legend(xticks);
xlabel('Trials', 'FontSize', lbsz);
ylabel('Average correct', 'FontSize', lbsz);


figure(2);
colormap parula;
b = bar(pfreq','hist')
%b(2).FaceColor = [0.25, 0.4, 0.1];
set(gca,'XTickLabel',xticks,'fontsize',axsz);
legend(xticks,'Location','northwest');
ylabel('Percent of total', 'FontSize', lbsz);




%%
%TRIAL 6 - equally sized distribtutions
%Extract ratings
ratings = data(:,1);
```

```
data2 = data(:,2:end);

ratings = floor(ratings);

%%
sind = zeros(1,9);
eind = zeros(1,9);
len = zeros(1,9);
for i = 1:9
temp = find(ratings==i);
sind(i) = temp(1);
eind(i) = temp(end);
if i == 1
len = numel(temp);
end
end

tlen = floor(0.8*len);
clen = len - tlen;

trat = [];
crat = [];
for i = 1:9
trat = [trat; i*ones(tlen,1)];
crat = [crat; i*ones(clen,1)];
end
%%
nloop = 500;
correct = zeros(9,nloop);
freq = zeros(9);
tic;
for count = 1:nloop
train = [];
test = [];
for i = 1:9
perm = randperm(len);
train = [train; data2(perm(1:tlen)+sind(i),:)];
test = [test; data2(perm(tlen+1:end)+sind(i),:)];
end

nb = fitcnb(train, trat, 'DistributionNames', 'mn');

pre = nb.predict(test);

correct(1, count) = sum(pre(1:clen) == crat(1:clen))/clen;
for i = 2:9
correct(i, count) = sum(pre((i-1)*clen:i*clen) ==...
crat((i-1)*clen:i*clen))/clen;
end
```

```
for i = 1:9
if i == 1
sind_t = 1;
else
sind_t = (i-1)*clen;
end
eind_t = i*clen;
for j = 1:9
freq(i,j) = freq(i,j) + numel(find(pre(sind_t:eind_t) == j));
end
end


if mod(count,10) == 0
toc;
count
tic;
end
end
toc;

%%
corr_avg=zeros(9,nloop);
for i = 1:nloop
for j =1:9
corr_avg(j,i) = sum(correct(j,1:i))/numel(correct(j,1:i));
end
end

pfreq = zeros(size(freq));
for i = 1:9
pfreq(i,:) = freq(i,:)/(nloop*clen);
end


%%
%TRIAL 7 - equally sized distribtutions without ones
%Extract ratings
ratings = data(:,1);
data2 = data(:,2:end);

ratings = floor(ratings);

%%
sind = zeros(1,8);
eind = zeros(1,8);
len = zeros(1,8);
for i = 2:9
```

```
temp = find(ratings==i);
sind(i-1) = temp(1);
eind(i-1) = temp(end);
if i == 2
len = numel(temp);
end
end

tlen = floor(0.8*len);
clen = len - tlen;

trat = [];
crat = [];
for i = 1:8
trat = [trat; i*ones(tlen,1)];
crat = [crat; i*ones(clen,1)];
end
%%
nloop = 500;
correct = zeros(8,nloop);
freq = zeros(8);
tic;
for count = 1:nloop
train = [];
test = [];
for i = 1:8
perm = randperm(len);
train = [train; data2(perm(1:tlen)+sind(i),:)];
test = [test; data2(perm(tlen+1:end)+sind(i),:)];
end

nb = fitcnb(train, trat, 'DistributionNames', 'mn');

pre = nb.predict(test);

correct(1, count) = sum(pre(1:clen) == crat(1:clen))/clen;
for i = 2:8
correct(i, count) = sum(pre((i-1)*clen:i*clen) ==...
crat((i-1)*clen:i*clen))/clen;
end

for i = 1:8
if i == 1
sind_t = 1;
else
sind_t = (i-1)*clen;
end
eind_t = i*clen;
for j = 1:8
```

```
freq(i,j) = freq(i,j) + numel(find(pre(sind_t:eind_t) == j));
end
end


if mod(count,10) == 0
toc;
count
tic;
end
end
toc;

%%
corr_avg=zeros(8,nloop);
for i = 1:nloop
for j =1:8
corr_avg(j,i) = sum(correct(j,1:i))/numel(correct(j,1:i));
end
end

pfreq = zeros(size(freq));
for i = 1:8
pfreq(i,:) = freq(i,:)/(nloop*clen);
end


%%
```

Python Code to extract and pair the ratings from the IMDb files to the plot follows:

```
import re
import sys

ratin = open(sys.argv[1],'r')
plotin = open('plottest','r')

fout = open('extracted_plots.txt','w')

done = object()
pline = iter(plotin)

names = {}
```

```
for rline in ratin:
t_split = re.split('::', rline.strip('\n'))
print(t_split)
names[t_split[1]] = t_split[0]

for line in plotin:
if 'MV:' in line:
pname = re.split('MV: ', line)[1].strip('\n')
if pname in names:
print(pname)
fout.write(pname + '::' + names[pname] + '::')
while '-------------------------------------------------------------------------------' not in l
if 'PL:' in line:
fout.write(re.split('PL: ', line)[1].strip('\n') + ' ')
line = plotin.next()
continue
fout.write('::')
line = plotin.next()
fout.write('\n')

ratin.close()
plotin.close()
fout.close()
```

Python Code to extract the movie titles and the ratings follows:

```
import re
import sys

filein = open(sys.argv[3],'r')
fileout = open('extracted_ratings.txt', 'w')

for line in filein:
linelist = re.split(' +',line)
name = linelist[4]
for part in linelist[5:]:
name += ' ' +  part
if float(linelist[2]) >= 1000:
if float(sys.argv[1]) <= float(linelist[3]) <= float(sys.argv[2]):
fileout.write('%s::%s' % (linelist[3], name))

filein.close()
fileout.close()
```