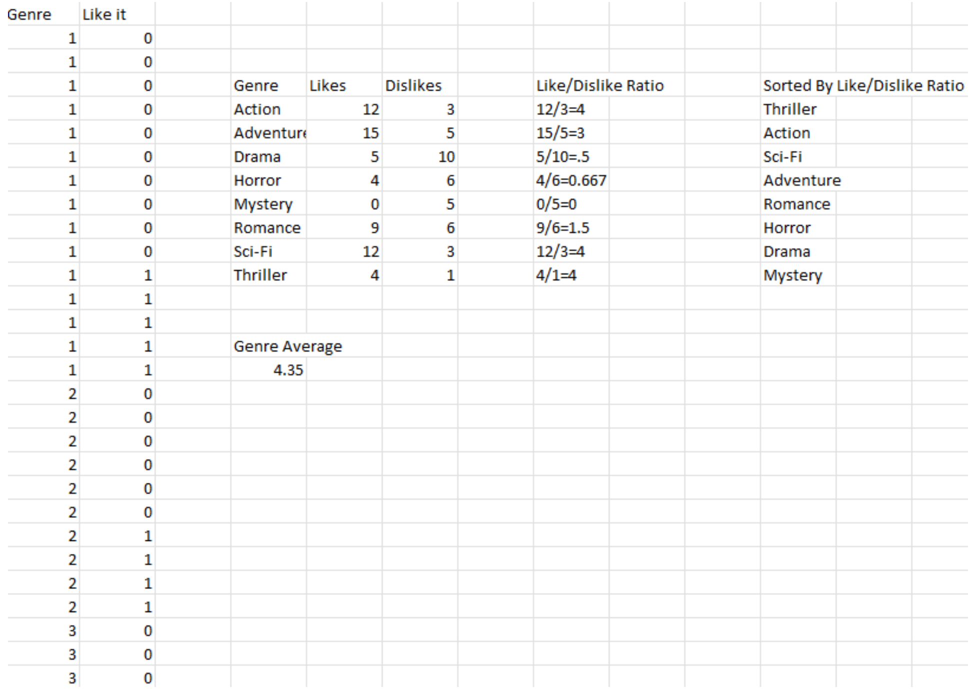
CS259 Assignment - Group 29

We first approached this problem by analysing the areas of the original student's model that were lacking and systematically improving upon them. The first of which was the missing vector operations, which are needed to perform calculations on our data. After we added these, we noticed that the knnClassify method was comparing the data, but not updating it. Instead, it was just asserting false which would result in the program failing without any nearest values being found. During our first iteration we replaced this assert statement with code that updated bestSimilarity and bestMatch to the respective current values if they were greater than the previous best values. This change resulted in our knnClassify method correctly finding the k-nearest neighbour of a value.

We then noticed that student X's program uses the movie id as a feature; this damaged their accuracy as id is a unique attribute. On our first iteration we amended this by using one-hot encoding for the genre attribute in its place as it is categorical. This immediately increased our accuracy from just 50% to 64%. Using a log10 transformation on runtime alongside the one-hot encoded genre boosted KNN's accuracy to 67% at k=1! (although it did drop with larger k values) After later discussion, we updated KNN to support any k value then implemented a genre list sorted by highest like/dislike ratio within the data set and an average transformation. To find the best value for k we calculated the square root of the dataset size and +/- 1 to ensure it is an odd number. With a dataset of 100 entries this resulted in k=11. These changes resulted in an accuracy of 67% at both k=1 and k=11.



Genre	Like it
1	0
1	0
1	0
1	0
1	0
1	0
1	0
1	0
1	0
1	0
1	1
1	1
1	1
1	1
1	1
2	0
2	0
2	0
2	0
2	0
2	0
2	1
2	1
2	1
2	1
3	0
3	0
3	0

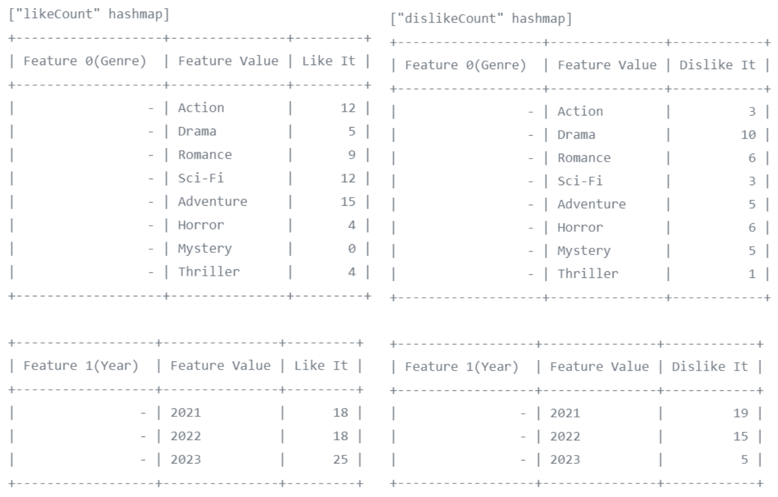
Genre	Likes	Dislikes	Like/Dislike Ratio	Sorted By Like/Dislike Ratio
Action	12	3	12/3=4	Thriller
Adventur	15	5	15/5=3	Action
Drama	5	10	5/10=.5	Sci-Fi
Horror	4	6	4/6=0.667	Adventure
Mystery	0	5	0/5=0	Romance
Romance	9	6	9/6=1.5	Horror
Sci-Fi	12	3	12/3=4	Drama
Thriller	4	1	4/1=4	Mystery

Genre Average
4.35

Our next move was to implement a simple probability model. This model compares movie feature values from the test dataset with the training dataset and if the values are the same it increments either a 'liked' or 'disliked' counter depending on the data. It then finds and compares the probabilities. This model resulted in 73% accuracy, making it our most accurate model however it is not the best option as the dataset values *must* be equal otherwise the program will just consider the movies as not being similar and basically fail the prediction. For example, comparing feature arrays [0, 4] and [1, 4] - even though they both have the same value for the second element, the program will just ignore it as the arrays must be exactly equal ([0, 4] != [1, 4]).

The next model we implemented was a version of the naïve bayes formula. From our testing we decided that the best feature to use with this model was ‘genre’ and this granted us 64% accuracy.

The first part of the code to implement this model makes tables using HashMaps, for example let's use genre and year as our features:



```
["likeCount" hashmap]                                ["dislikeCount" hashmap]
+------------------+--------------+---------+        +------------------+--------------+------------+
| Feature 0(Genre)  | Feature Value | Like It |       | Feature 0(Genre)  | Feature Value | Dislike It |
+------------------+--------------+---------+        +------------------+--------------+------------+
|                 - | Action       |      12 |       |                 - | Action       |          3 |
|                 - | Drama        |       5 |       |                 - | Drama        |         10 |
|                 - | Romance      |       9 |       |                 - | Romance      |          6 |
|                 - | Sci-Fi       |      12 |       |                 - | Sci-Fi       |          3 |
|                 - | Adventure    |      15 |       |                 - | Adventure    |          5 |
|                 - | Horror       |       4 |       |                 - | Horror       |          6 |
|                 - | Mystery      |       0 |       |                 - | Mystery      |          5 |
|                 - | Thriller     |       4 |       |                 - | Thriller     |          1 |
+------------------+--------------+---------+        +------------------+--------------+------------+

+-----------------+--------------+---------+          +-----------------+--------------+------------+
| Feature 1(Year)  | Feature Value | Like It |         | Feature 1(Year)  | Feature Value | Dislike It |
+-----------------+--------------+---------+          +-----------------+--------------+------------+
|                - | 2021         |      18 |         |                - | 2021         |         19 |
|                - | 2022         |      18 |         |                - | 2022         |         15 |
|                - | 2023         |      25 |         |                - | 2023         |          5 |
+-----------------+--------------+---------+          +-----------------+--------------+------------+
```

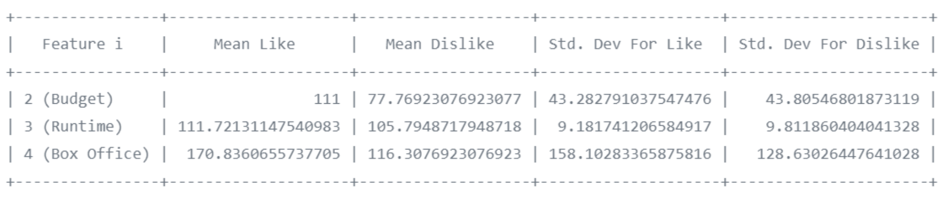
We then use these values to calculate the probabilities using the naïve bayes formula:

$$P(a|b) = \frac{P(b|a) \cdot P(a)}{P(b|a) \cdot P(a) + P(b|\neg a) \cdot P(\neg a)}$$

Our method utilises HashMaps to make things easier for making the formula support multiple features instead of only one.

Finally, we discussed as a group how best to improve upon this model and came to the conclusion that, as the dataset has many continuous variables (e.g. runtime, imdb, rt, budget, box office) it would be appropriate and even beneficial to implement a gaussian naïve bayes model.

For continuous features we use a HashMap and a double array of 4 elements to make a table of all the means and standard deviations, for example; using budget, runtime, boxOffice.

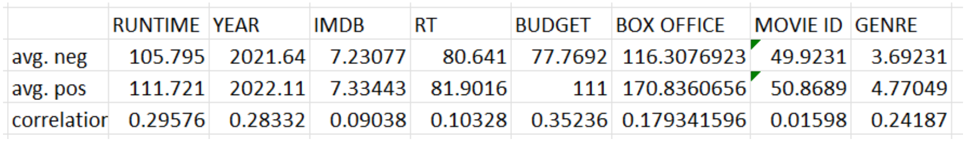


```
+----------------+--------------------+-------------------+--------------------+---------------------+
|   Feature i    |     Mean Like      |   Mean Dislike    | Std. Dev For Like  | Std. Dev For Dislike |
+----------------+--------------------+-------------------+--------------------+---------------------+
| 2 (Budget)     |                111 | 77.76923076923077 | 43.282791037547476 |    43.80546801873119 |
| 3 (Runtime)    | 111.72131147540983 | 105.7948717948718 |  9.181741206584917 |    9.811860404041328 |
| 4 (Box Office) |  170.8360655737705 | 116.3076923076923 | 158.10283365875816 |   128.63026447641028 |
+----------------+--------------------+-------------------+--------------------+---------------------+
```

These values are then used in the gaussian naïve bayes model, which is essentially the same as the naïve bayes model but uses an extra formula in the process for continuous features:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

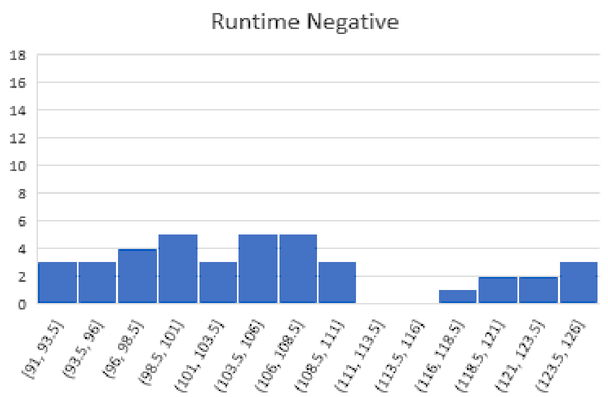
Using this approach with the features that had the highest correlation from excel we were able to get a 68% accuracy in the gaussian naïve bayes model, a 4% accuracy increase from the classical naïve bayes' model.

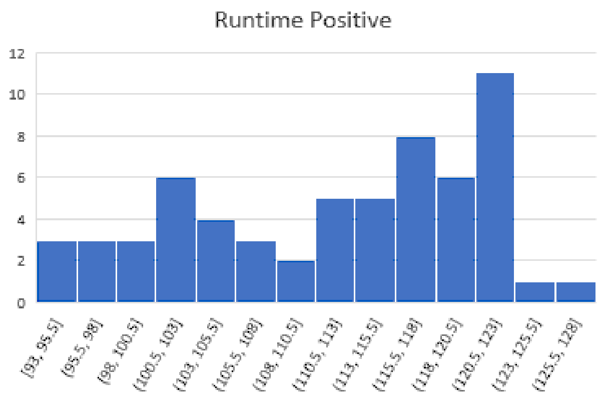


	RUNTIME	YEAR	IMDB	RT	BUDGET	BOX OFFICE	MOVIE ID	GENRE
avg. neg	105.795	2021.64	7.23077	80.641	77.7692	116.3076923	49.9231	3.69231
avg. pos	111.721	2022.11	7.33443	81.9016	111	170.8360656	50.8689	4.77049
correlatior	0.29576	0.28332	0.09038	0.10328	0.35236	0.179341596	0.01598	0.24187

In excel, we found the correlations and averages for every feature. Movie id has a very low correlation, which links back to how bad it is to use for predicting - like student X has for KNN.

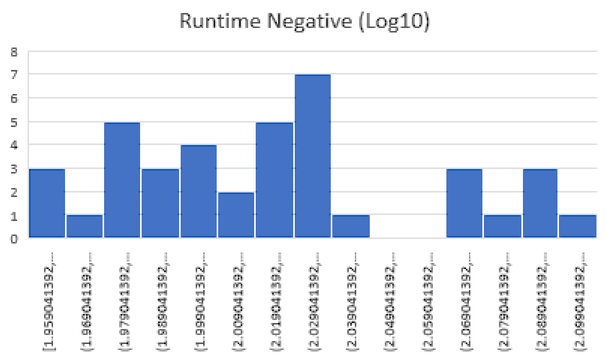
We decided to pick genre, runtime, year, budget and box office to use for the gaussian naïve bayes model as they all have high correlations compared to the others.

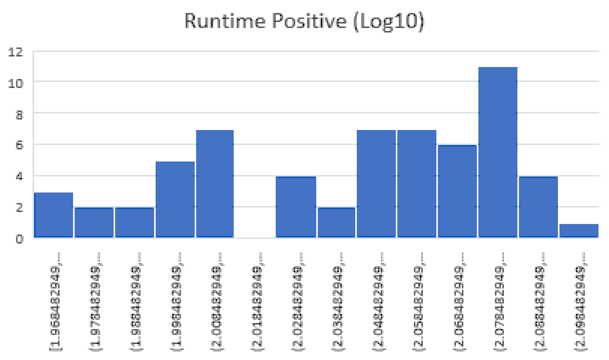




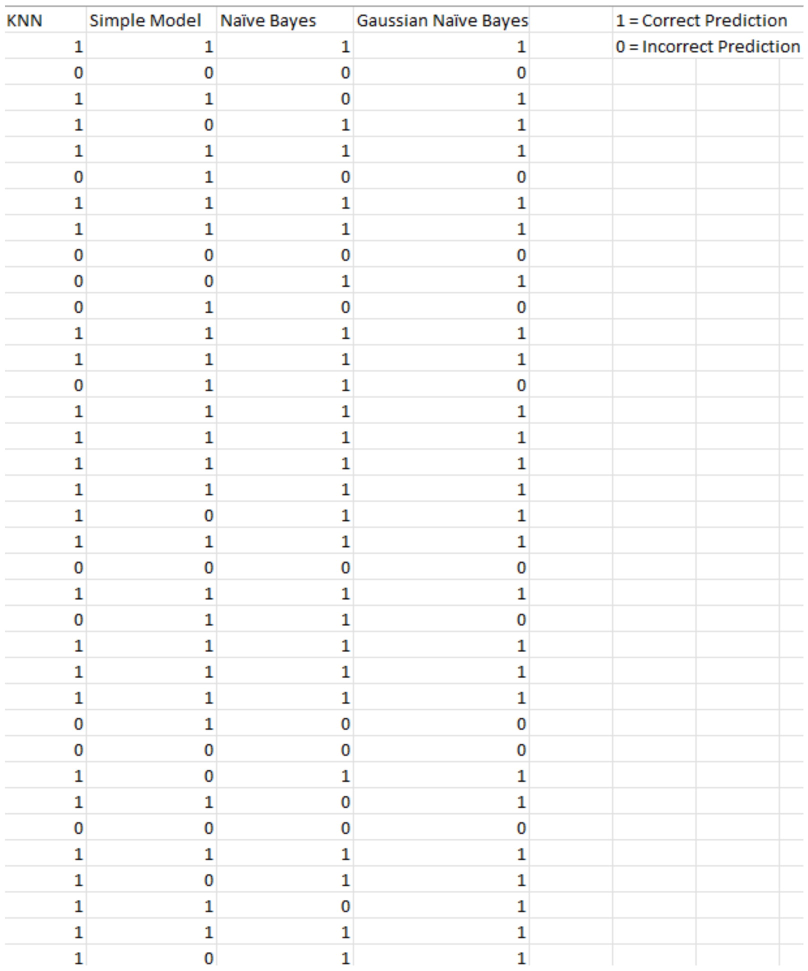
Some of the features we chose had good histograms, such as runtime. The 'runtime positive' graph had higher values than the 'runtime negative' histogram, which was indeed useful for gaussian naïve bayes.

However, the 'runtime positive' histogram doesn't look "normal" (gaussian) so it was not too useful for KNN. Although, some of the other features we chose for it at first, did look "normal".





Applying a log10 transformation for the runtime made the histograms look more gaussian, which gave good results in KNN k=1, as we said previously.



KNN	Simple Model	Naïve Bayes	Gaussian Naïve Bayes	1 = Correct Prediction
1	1	1	1	0 = Incorrect Prediction
0	0	0	0	
1	1	0	1	
1	0	1	1	
1	1	1	1	
0	1	0	0	
1	1	1	1	
1	1	1	1	
0	0	0	0	
0	0	1	1	
0	1	0	0	
1	1	1	1	
1	1	1	1	
0	1	1	0	
1	1	1	1	
1	1	1	1	
1	1	1	1	
1	1	1	1	
1	0	1	1	
1	1	1	1	
0	0	0	0	
1	1	1	1	
0	1	1	0	
1	1	1	1	
1	1	1	1	
1	1	1	1	
0	1	0	0	
0	0	0	0	
1	0	1	1	
1	1	0	1	
0	0	0	0	
1	1	1	1	
1	0	1	1	
1	1	0	1	
1	1	1	1	
1	0	1	1	

At the end, comparing what each model got correct/incorrect gave us more information. We noticed that at some cases, all the models give an incorrect prediction – which we think is most likely due to some missing training data which made the models guess wrong.

At other cases we notice that usually only at most 2 were giving an incorrect prediction. Perhaps combining the models could give a better accuracy? We tested this by incrementing an index when at least one of the models predicts correctly and then calculating the accuracy at the end, which gave us 86% in total.