DCU School of Computing
Assignment Submission

Student Names:	Kyrylo Khaletskyy, Ben Kelly (Group 11)
Student Numbers:	15363521, 15337716
Programme:	BSc. in Computer Applications
Project Title:	Predicting Box Office Revenue - CA4010 Report
Module code:	CA4010
Lecturer:	Mark Roantree
Due Date:	17/12/18

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged, and the source cited are identified in the assignment references.

I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.

This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at http://www.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines.

I understand that I may be required to discuss with the module lecturer/s the contents of this submission.

Signed: Kyrylo Khaletskyy, Ben Kelly

Date: 9/12/2018

1. Introduction

The following report details how we used our Kaggle dataset of IMDB movie data to predict what the gross domestic revenue of a certain movie will be to an accuracy of 56.07**%.** We used the following technologies to achieve this with data mining skills gained from the CA4010 module this semester.

- Python
- Scikit-learn
- Matplotlib & plot.ly
- Microsoft Excel & Numbers for Mac

The report will also disclose any judgments we made regarding our data and insights gained from the entire process.

2. Idea & Dataset Description

We used a dataset of movies from 1916 to 2016, which included just over 5000 entries with 28 different attributes (see link below for more information on our dataset). Our idea was to combine some or all of these attributes to observe if it was possible to predict how financially successful a movie would be given a subset of these attributes. We had to decide what was important to a successful movie or in contrast what goes wrong with unsuccessful movies. How important are factors like having accomplished actors or well-known directors involved in the movie or does the IMDB rating of a film correlate to its box office revenue? We had to answer all these questions through our dataset and calculations.

As was expected we could not use the entirety of this dataset. We will talk about this further in the Data Preparation section of the report. Originally we cut too many attributes out, assuming they were not relevant, this was naive of us and we later calculated our prediction based on as many numerical/categorical data as possible. Out of the 28 attributes we were given we decided to use 7 for prediction purposes. Which will be quickly defined later.

Link to dataset → https://www.kaggle.com/suchitgupta60/imdb-data

3. Data Preparation

3.1 Formatting Data

Although our dataset was taken from Kaggle the Data was extremely messy and there were numerous missing fields and null values. To alleviate some of these issues we began to format our data.

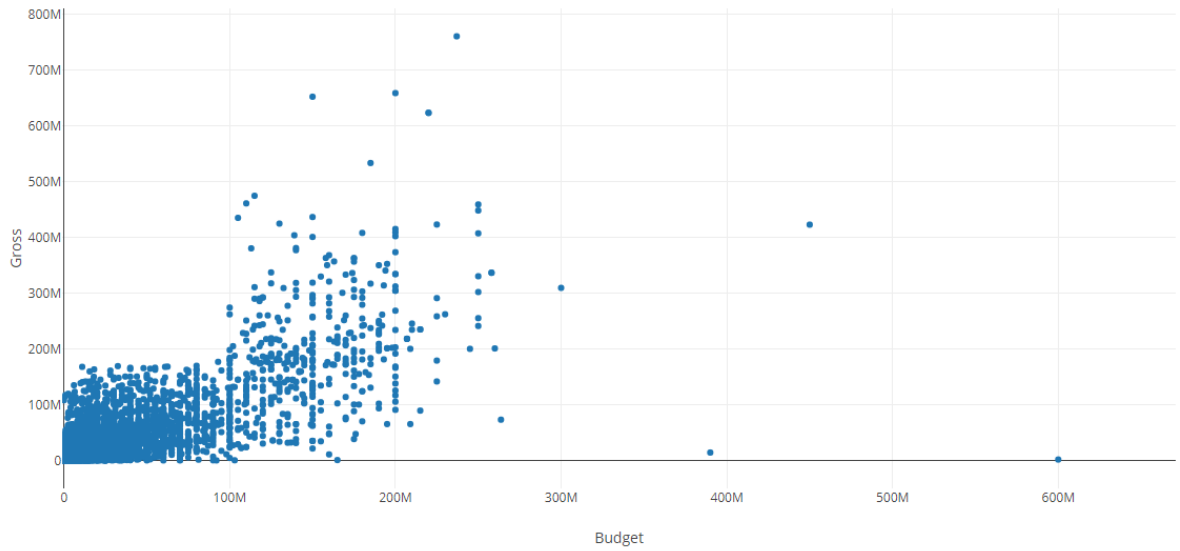
As we were looking to calculate gross revenue, any entries that were empty (null) for this attribute had to be discarded as they cannot be used for the calculation. These were 844 entries unfortunately which is a big portion of our dataset, taking us from 5044 entries to 4160.

```
def ReadIn(filename):
    num = 0
    lines = []
    with open(filename, 'r') as f:
        for line in f.readlines():
            if num >=1:
                line = line.strip("\n").strip("\r")
                name = line.split(';')
                lines.append(name)
            num += 1

    return lines
```

```
def RemoveNulls(csv_file):
    for row in csv_file:
        for i, x in enumerate(row):
            if len(x) < 1:
                x = row[i] = 0

    return csv_file
```

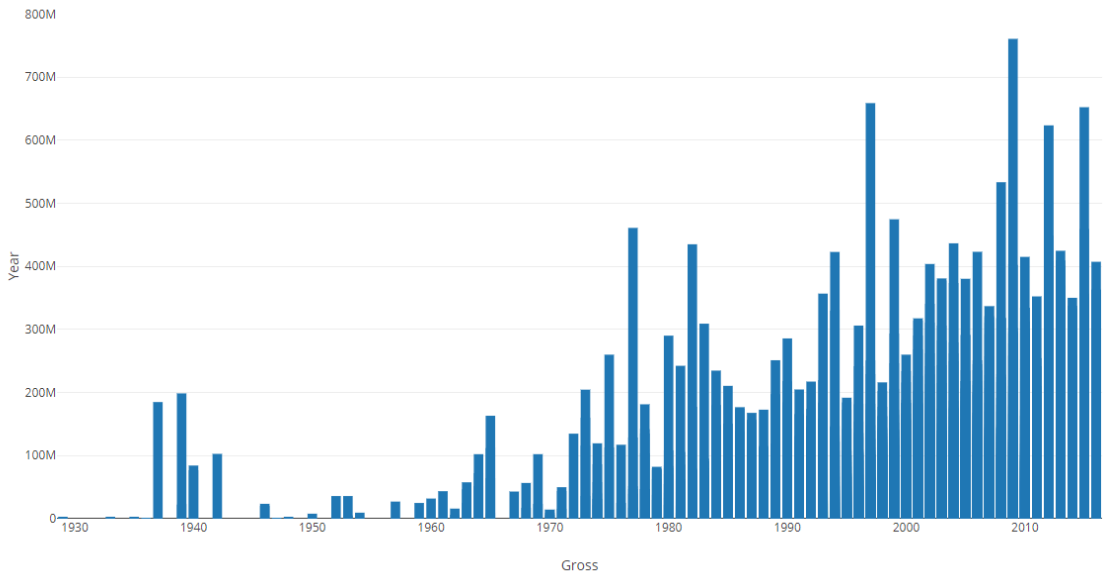
There were non-alphanumeric values at the end of each movie title which we removed for better visualisation. Commas within certain cells would interfere with the split when calculating values so they were removed. Useless values like newline characters and return characters were also taken out. When calculating numerical attributes such as null values would reduce our accuracy. A workaround we opted for was replacing null values with 0. This way we could easily calculate each numerical attribute without becoming less accurate with each null value encountered. Within our dataset, there were 45 duplicate rows so these were also removed.

As there were 28 attributes for each movie some of these would not be useful to us for calculation purposes. We removed obvious non-important attributes when we were sure they had no impact on the revenue a movie made. Attributes such as whether the movie was in colour or not seemed useful initially but only 100 of these values were black and white.

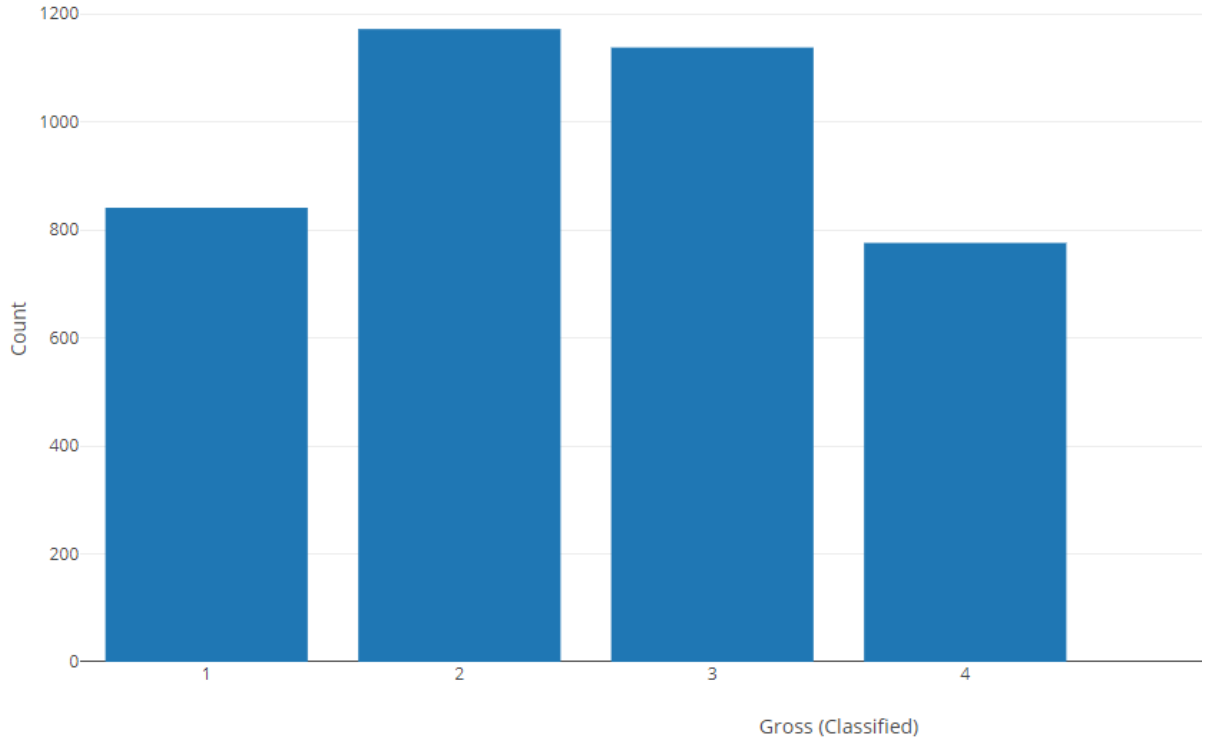


Upon graphing other attributes, extreme outliers were observed some values for budget(shown above) and cast_likes were 2 to 3 times the average. This graph shows the few outliers we are talking about for budget. To combat inaccuracies these were removed from the dataset.

3.2 Classifying & Normalising Data



We were looking to calculate gross domestic revenue for a given movie. We decided after we graphed our data initially that it would be necessary to classify our data. We looked at our graph of revenue to get insights into our data, particularly where boundaries might lie. Graphing the data by year we got useful insights into how the movie industry has become popularised overtime with huge revenues being achieved in the last twenty years.



```
def Classify(gross):
    i = 0
    for item in gross:
        if item <= 5000000:
            gross[i] = 1
        elif item> 5000000 and item <= 30000000:
            gross[i] = 2
        elif item > 30000000 and item <= 80000000:
            gross[i] = 3
        elif item > 80000000:
            gross[i] = 4
        i+=1;
    return gross
```

Gross revenue ranged from $162 to $760,505,847. We classified them into 4 categories (1 - 4). 4 being extremely successful and 1 being a flop. This gave us the following split graphed above. 841 in category 1 (poor), 1172 in category 2 (unsuccessful) and 1138 in class 3 (good) and 776 in class 4 (extremely successful). The reason we split is because we couldn't calculate completely continuous data and it would give us extremely low accuracy, in other words trying to calculate a value between $162 to $760,505,847 is futile. We decided to split the data into 4 categories, because it can be more than good or bad (binary), these classes were split in a ratio shown above because majority of our revenues lie between class 2 and 3, additionally it should be harder to obtain a score of 1 or 4 as they are the worst and best case scenario.

We then partitioned the set using a ratio of 85:15, as a good proportion of the dataset was lost due to ill-formatted data or null values that we couldn't use e.g. in gross revenue. So we decided that a small amount more of training data was needed compared to the 80:20 standard.

```
def NormCalc(x, min_val, max_val):
    return (x - min_val) / (max_val - min_val)


def Normalise(csv_file):
    values = []
    gross = []

    for line in csv_file:
        for i in range(2,10):
            line[i] = float(line[i])
        for j in range(3,10):
            line[j] = NormCalc(line[j], MinCalc(j, csv_file), MaxCalc(j, csv_file))

        values.append(line[3::])
        gross.append(line[2])

    return values, gross
```

$$\frac{a - min}{max - min}$$

In order to use our attributes, we had to normalise them. That way not only could we use it but it would also make it easier to weight the data when we need to. This was done by using the formula given in the CA4010 notes. This gave us a value from 0 - 1 for all of the numerical attributes we were using as part of the calculation. The implementation of this function can be seen in the screenshot above.

3.3 Attribute Selection

We quickly discovered that if a certain movie had a large following on Facebook (director, top 3 actors, cast in total) the higher the revenue was in general. To make use of this correlation we decided that a combination of these values would be optimal for deciding if a

movie had a good following that way if a movie had a low amount of likes for the director but high values for the total cast and top 3 actors it wouldn't be skewed. Below is a list of attributes which we used in our calculation. We also created a new column called social_media, which is explained in results and analysis.

- **Director_likes:** The number of likes the director of the movie has on Facebook
- **Actor_likes:** The number of likes the top three actors have on Facebook combined
- **Cast_likes:** The number of likes the entire cast have on Facebook
- **Social_media:** A score assigned to each movie by combining Director_likes + Actor_likes + Cast_likes and categorised (0 - 10)
- **Facenumber:** The number of people shown in the poster
- **IMDB:** The score a movie received on IMDB.com (1 - 10)
- **Budget:** The budget of the movie in Dollars

4. Algorithm Description

After our workshop, we decided that categorizing our data would be necessary due to the low correlation between extremely continuous attributes. This limited the selection of algorithms we could use down to Naive Bayes, KNN and Decision trees.

```
csv_file = ReadIn('imdb_data.csv')
csv_file_parsed = RemoveNulls(csv_file)
values, gross = Normalise(csv_file_parsed)
gross = Classify(gross)

print "split:", SplitCount(gross)

# Naive Bayes
values_train, values_test, gross_train, gross_test = train_test_split(values, gross, test_size = .15, random_state = 20)

#Bernoulli
BernNB = BernoulliNB()
BernNB.fit(values_train, gross_train)
gross_expect = gross_test
gross_predict = BernNB.predict(values_test)
print "BernoulliNB:", accuracy_score(gross_expect, gross_predict)

#Gaussian
Gauss = GaussianNB()
Gauss.fit(values_train, gross_train)
gross_expect = gross_test
gross_predict = Gauss.predict(values_test)
print "GaussianNB:", accuracy_score(gross_expect, gross_predict)

#Multinomial
Multi = MultinomialNB()
Multi.fit(values_train, gross_train)
gross_expect = gross_test
gross_predict = Multi.predict(values_test)
print "MultinomialNB:", accuracy_score(gross_expect, gross_predict)
```

When choosing an algorithm to implement we decided that describing our data would help us make decisions. We came to the conclusion that our dataset is relatively small with 3876 rows**,** with high bias and skewed towards the larger end of the revenue scale. The majority of our attributes are not formatted in the way that decision trees would be useful so that was almost immediately eliminated. SKLearn has the ability to use three different types of Naive Bayes (Bernoulli, Gaussian and Multinomial) so we decided to go with that. This way we could use three calculations, pick the best and observe why it performed best and what the differences were, be it different attributes, weights and trimming.

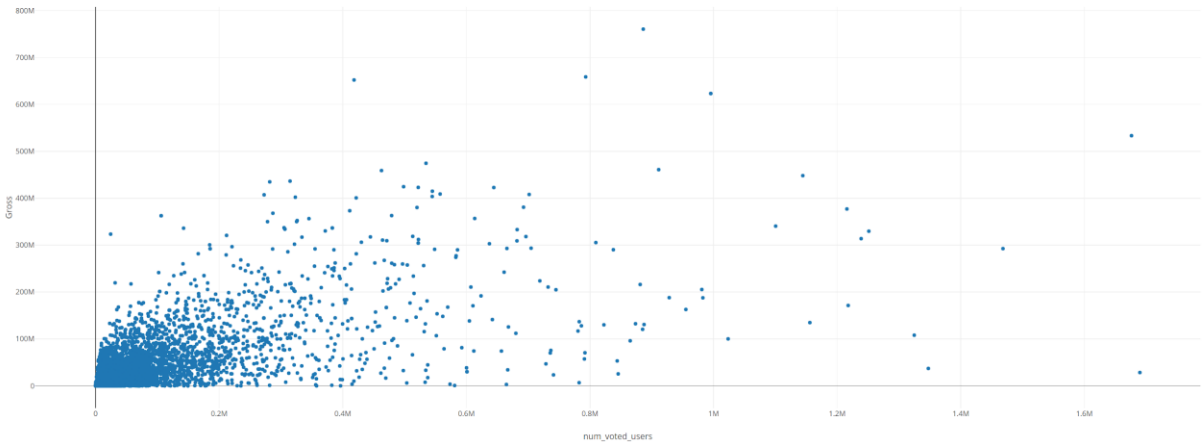
Mark pointed out in our workshop that if there is a strong correlation between certain actors and directors always working together this correlation would decrease our accuracy. Naive Bayes doesn't take these potential correlations/biases into account as it assumes

every attribute is conditionally independent of one another. Another issue we faced was trying to implement attributes like director and actors into our algorithm. There were 2400 unique directors of movies and well over 5000 unique actors so classifying these attributes would have been impossible. A workaround we came up with was using the director's likes on Facebook and the top three actors likes on Facebook, this would give us a numerical representation of how popular specific directors, actors and the combination of the two were.

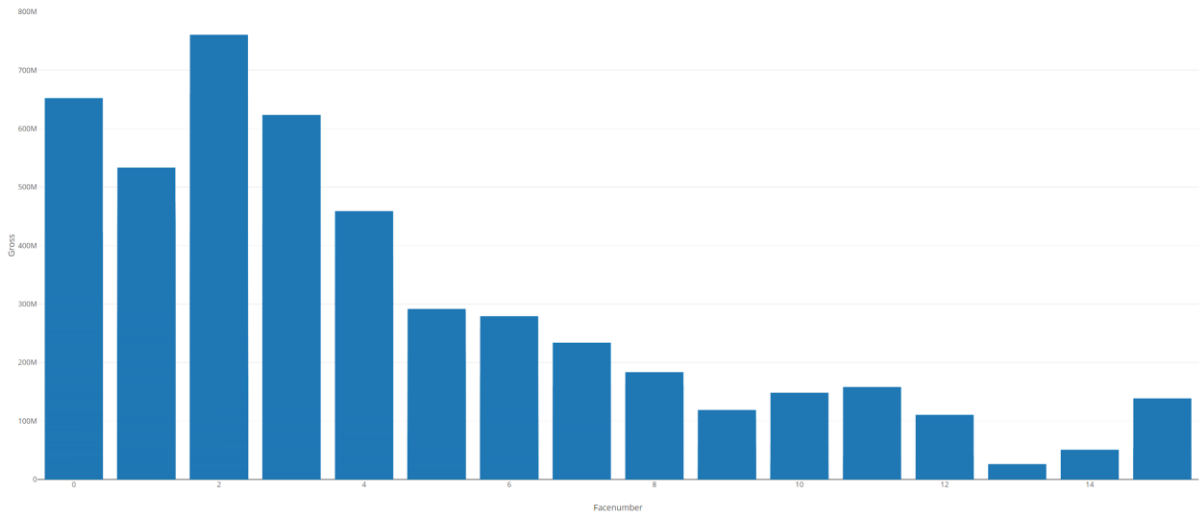
```
['director_likes', 'actor_likes', 'cast_likes', 'social_media', 'facenumber', 'imdb', 'budget']
[0.3157894736842105, 0.04261221374045802, 0.08349550043396829, 1.0, 0.13333333333333333, 0.89, 0.395]
[0.2947368421052631, 0.01984732824427481, 0.02277800618214487, 0.9, 0.13333333333333333, 0.62, 0.30833333333333335]
[0.25263157894736843, 0.017219847328244275, 0.018197737274070015, 0.8, 0.26666666666666666, 0.73, 0.15]
[0.0008842105263157895, 0.0528030534351145, 0.05353950634202793, 1.0, 0.0, 0.67, 0.24833333333333332]
[0.0, 0.02119389312977099, 0.02239276414965054, 0.8, 0.06666666666666667, 0.5900000000000001, 0.08333333333333333]
```

After all the data was formatted, classified and normalised and put into lists that can be used by our algorithm (items index, 0, 50, 500, 1000, 2000 in the screenshot above), we then ran it through three different Naive Bayes algorithms, these were Bernoulli, Gaussian and Multinomial. While calculating accuracies we began weighting our attributes on what we believed to be relevant. We implemented a function which would further multiply our normalised values by (0.1 - 1) with the lower multiplier believed to not be relevant and higher multiplier believed to be most relevant. We soon discovered that any sort of weighting would bring our accuracy down, this makes sense as Naive Bayes in SKlearn implements it's own weighting system so any multiplication down on our end would be detrimental to the work of the algorithm. Results for each algorithm can be found in the results and analysis section below.

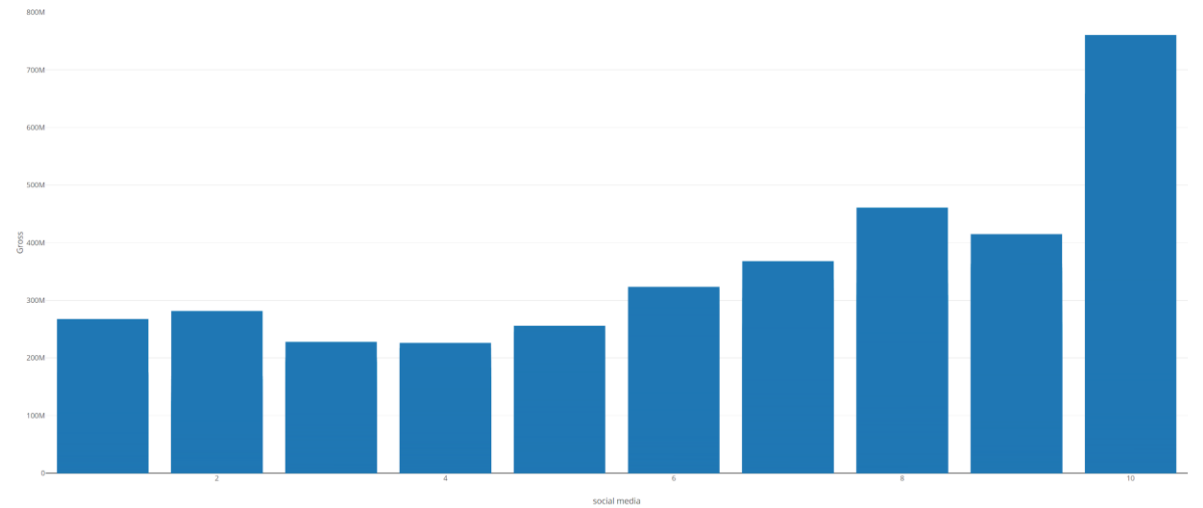
5. Results & Analysis



In the beginning, there were rows which we initially thought would be used for our calculation. Number_of_voted_users initially seemed useful but we found that there is no strong correlation between it and gross, as movies with a large amount of reviews/comments don't necessarily mean that they are good/bad reviews as shown in the screenshot above. Movies with both a high and low gross revenue had a high number of reviews. When using these attributes the accuracy went down as much as 10% over the three algorithms.

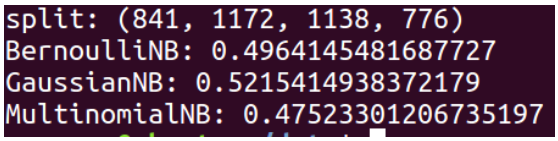


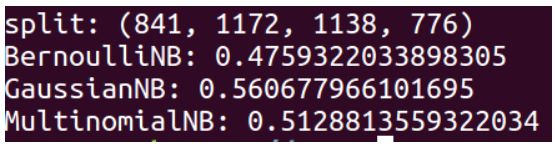
The above graph shows the revenue based on the number of faces in a poster. This is an interesting finding as this graph is skewed by mean averages taking in large figures for gross on the usual range of ‘facenumbers’ in a poster (0-4). As the number of people increase in the poster the fewer occurrences we have for this, thus this graph is skewed left. Based on this we found that movies with up to 4 faces in a poster generally have a higher revenue than movies with more than 4 faces in a poster.



Another set of fields which we overlooked initially were the facebook likes of actors, directors and cast. We didn't realise that older movies wouldn't have director and actor facebook profiles and therefore would be missing values for “likes”. To combat this we created a new column called “social_media” which scored the aforementioned attributes. The facebook like columns was added together (this also reduced the number of cells which had a value of 0) and set to a score between 0 and 10, which was then normalised to a value between 0 and 1. This gave us an accuracy increase of 4% on the Gaussian algorithm, 3.5% increase on the Multinomial algorithm, and 2% decrease on the Bernoulli algorithm. As shown in the screenshot above we can see that the new social_media score has a much higher positive correlation between gross revenue than director, actor and cast likes.

Without social_media:





With social_media:

The results from the various Naive Bayes algorithms are listed above. As we can see Gaussian Naive Bayes gives us the best accuracy. The reasons why our accuracy varies between types of NB algorithm is because:

Bernoulli NB: Works best when attributes are binary or boolean in type like True/False or 0/1. None of our attributes are in this form which is why it performs poorly. As shown in the screenshots above Bernoulli is the only algorithm to decrease in accuracy with the implementation of social_media score. This can be attributed to having fewer values to use within the calculation, Bernoulli usually works better with a larger amount of data attributes and also the fact that three of the columns which were between 0 - 1 have now been replaced by one.

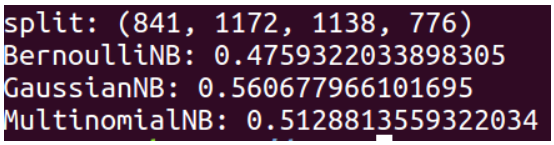
Gaussian NB: Works best when attributes are in decimal form. GNB assumes features to follow a normal distribution. As our data is split into four classes which are slightly normal in terms of distribution i.e lower count of values on the extremes and high count for middle data. Gaussian has the biggest increase of accuracy when we made the change. This can be attributed to having fewer values of zero but also by having fewer attributes to check, as Gaussian NB is quite a simple classifier it can sometimes work better with fewer columns.

MultiNomial NB: Works best when attributes are discrete values like word count 1,2,3 etc. Some of our data is in this format which is why it performs reasonably well. We believe there is a slight increase in the accuracy of the Bernoulli algorithm because with the new social_media algorithm there are fewer values that are 0 scattered throughout actor_likes, director_likes and cast_likes.

As mentioned in data preparation we split our gross revenue into 4 classes (1-4). Initially, we planned to class this data into 5 classes (1-5), but this gave us an accuracy decrease of 6% (excluding social_media from the calculation). Similarly, when we tried to class the data into 3 classes (1-3) this gave us an accuracy increase of 4% (excluding social_media from the calculation). We decided that 4 classes would be an ideal middle ground for our dataset as 3 would be too small for such a large difference in the lowest and highest gross revenue value, and 5 classes reduce our accuracy down to an undesired amount. We were also conscious of overfitting our classification algorithm so 'tinkering' of the data was kept to a minimum.

Although a result of 56% might seem low, given that we are looking to predict 4 different classes it suggests that our system has an accuracy that is more than twice as accurate as pure chance. While getting consistent values between 50% - 56% we began to wonder was there something inherently flawed within our dataset. We looked over how we prepared our data, the graphs of the data and what conclusions/assumptions we made. Was it simply that our dataset was too small? Did we include all useful attributes? Did we use the right algorithm? To answer the first question, no, our dataset was not too small but it would be favourable for it to be larger and contain 'fuller' data, i.e. fewer missing values.

Final Result



The final results after all of our efforts experimenting with different attributes, classification brackets and creating new attributes based on other attributes had led us to a final accuracy of 56.07% using the Gaussian Naive Bayes as shown in the screenshot above. The attached CSV file contains the new transformed data which we used in the calculation shown above. It contains all the attributes we needed along with the title of the movie and year. The gross revenue of the final CSV file is not classified and kept in its original form in order for us to be able to graph our results but can be easily classified within our python code and written to a new CSV file.

We maintain that Naive Bayes was the correct algorithm for our dataset, it gave us a lot of flexibility with our data and attributes remained conditionally independent. It also helped us when we were still deciding on what attributes to include as it is less sensitive to irrelevant features than other classifiers would be. Overall it gave us good insight into what was important within our data and gave us a good solution to our multi-class classification problem.

What we learned

At the beginning of the project, we naively made assumptions about our dataset. We almost immediately assumed that certain attributes weren't relevant without any real proof other than our intuition. After some deeper calculations and reading of the ca4010 notes we quickly realised that some of the attributes might actually be relevant to our calculation. For instance, we immediately assumed movies like Avatar and Titanic were outliers and that we should remove them so we proposed a trim of our dataset by 2% on both ends. When calculating we came to realise that extreme values for gross revenue actually helped prove our point and increased our accuracy. Needless to say this trim was reverted.

We were looking to calculate gross domestic revenue for a given movie, what we did not consider initially was that our dataset goes back as far as 1916, so $1 million was worth a lot more than the same amount by today's standards. If we were to implement this solution further it would be nice to calculate for inflation. Also Facebook was not invented until 2004

and did not become widely used until the early 2010s, therefore some of the methods that were used to collect data about our attributes were not perfect. This led us to two conclusions.

1. The gross will remain static over time (US Dollars $)
2. Facebook likes will be used as this still gives us a good indication of modern movie popularity and can split modern and pre-modern movies based on these values.

Although our initial dataset was over 5000 rows we could only use 3800 or so rows in our final implementation. Ideally, we would have liked to have a lot more data to train and test our system on, our advice for any students sitting CA4010 in the future would be to use a dataset as large as possible.