Movie predictive model overview

CS259: Quantitative Methods for Computer Science

Table of Contents

1. Introduction:

Predictive modelling is vital to modern data science for predicting future outcomes and making data-driven decisions (Igual et al., 2017). In Alex's case, accurate predictive models can help personalise the viewing experience by forecasting preferences based on a training dataset. By utilising machine learning algorithms such as K-nearest neighbour which identifies patterns by comparing the similarity of new data points to previously observed data, predictive models can effectively predict outcomes (Syriopoulos et al., 2023). This report focuses on the systematic enhancement of Alex's movie predictive model, which was initially implemented on a defective implementation of the KNN algorithm.

2. Initial model analysis:

The initial predictive model created by Alex demonstrates several technical and methodological shortcomings, which hinder its overall utility. The first issue with Alex's implementation was that it did not calculate the accuracy, this meant that the initial implementation did not compile. Another issue was that the genres stored in the features vector had different weightings. For example "Thriller" was given a weighting of seven whereas "Action" was weighted zero. This meant that when the accuracy calculation was implemented, "Action" would not be considered which would make predictions inaccurate. The model also used "Movie ID" as a feature, which provided no predictive value. The final drawback we discovered was that the KNN classifier was restricted to K equals one, meaning it was relying on the closest neighbour. This implementation only achieved 50% accuracy, which is the equivalent of random guessing.

2.1 Feature selection:

The feature selection process is important to improving the predictive model's accuracy. Through careful examination of both correlation metrics and data visualisations through the use of histograms, the most influential predictors were identified. The feature correlation analysis was central to this process, because it quantified the strength and direction of relationships between features, allowing us to identify key predictors that significantly influenced the accuracy of the model(Udovicic et al., 2007). From figure five we can see that features like "Budget", "Box Office", "IMDB" and "Runtime" exhibited strong positive correlations with Alex's preferences, and were therefore selected for the final model. In contrast, features such as "MovieID", "Title", "Director", and "Lead Actor" showed minimal correlation with Alex's preferences, leading to their exclusion in the final model. This strategic decision-making ensured that the models focused on the most informative features, improving their overall accuracy. The feature "Budget" provides a compelling example of the value of this analytical approach. The histograms displayed by figures one through four reveal distinct patterns in the distribution of budget values for both liked and disliked movies. When creating the histograms it was essential to have histograms also applied with Log10 to handle skewed data, allowing for better visualisation. We can recognise that the budget-negative histogram shows that Alex's dislikes tend to have a left-skewed distribution, with most of the disliked movies having a lower budget. In contrast, the

budget-positive histogram demonstrates a more balanced or right-skewed distribution, indicating that Alex's liked movies tend to have higher budgets.

Figure one

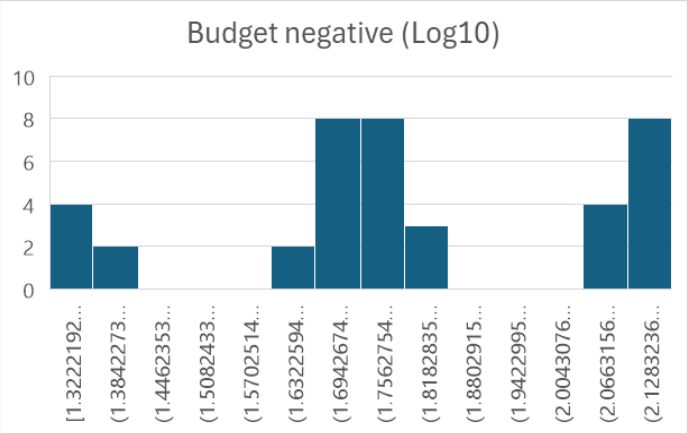
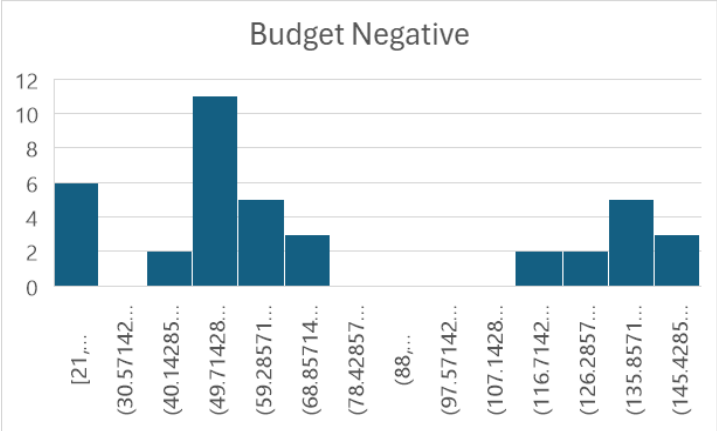


Figure two



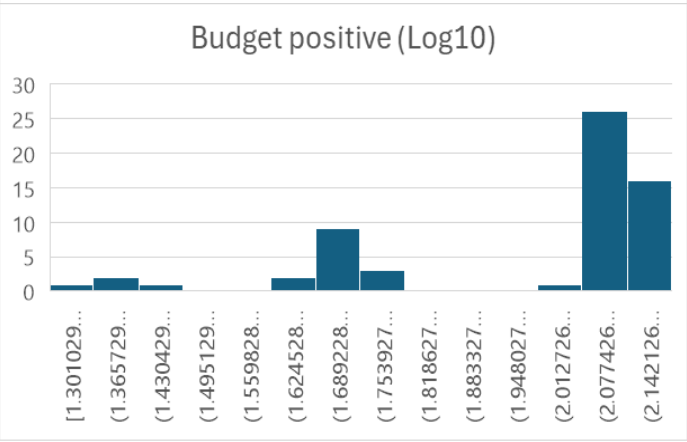


Figure three

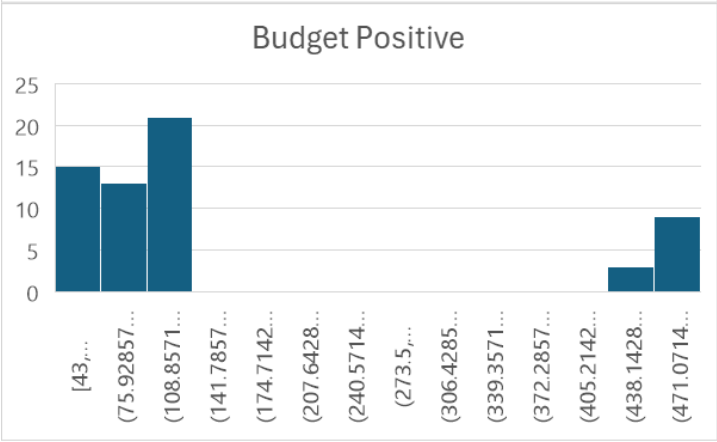


Figure four

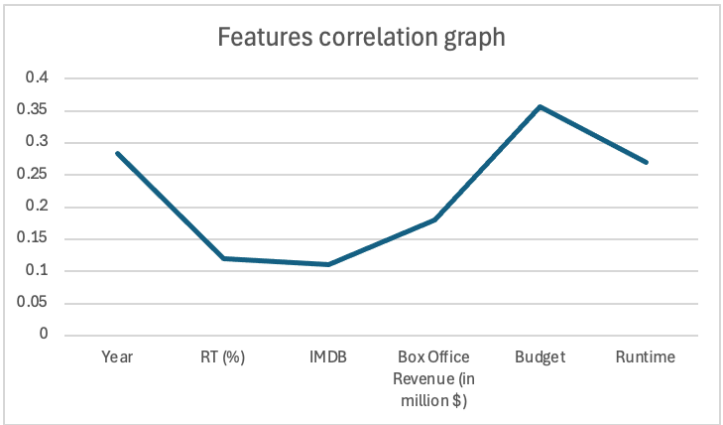


Figure five

2.2 Transformation techniques:

Transformation techniques were employed to improve the model's accuracy. One hot encoding was used for the genre feature, changing each genre into binary values. This approach enables the model to independently assess the impact of each genre on the accuracy, rather than relying on a non-numeric, weighted alternative(Seger, 2018). For numerical variables such as “IMDB”, “Budget”, and “Box Office” a feature weighting strategy was used to prioritise features with stronger correlations. Normalisation techniques such as Z-score were also applied to standardise feature ranges to a normal scale (e.g., [-1, 1]), mitigating the disproportionate influence of larger magnitude variables like “Budget” over smaller features like “IMDB”. These transformations led to immediate improvements in the model’s performance from the initial accuracy of 50%.

3. Model Implementation Approaches:

3.1 Model One:

In Model one, the primary issues with the template were addressed, particularly the weighting of the genre features. To resolve this, all genres were assigned equal weight, ensuring that no genre disproportionately influenced the model’s accuracy. Additionally, other features from the training dataset such as “Budget”, “Year”, “IMDB”, “Box Office”, “Rotten Tomato” and “Runtime” were incorporated into the features vector.

Initially, Alex’s implementation could only identify the nearest neighbour (K = 1). To address this, two vectors were created: one to store all the similarity values (calculated using the dot product) and another for the corresponding labels. We then used bubble sort to sort the similarities from most to least similar while mapping the labels to the sorted similarities. Depending on the value of K, we checked for the first K values in the labels array. If the majority were one, the movie was predicted to be liked; if zero, it was predicted to be disliked.

```
static int knnClassify(double[][] trainingData, int[] trainingLabels, double[] testFeature, int k) {  1 usage  Moad

    // Array to store the similarity values between the test feature and each training data point
    double[] similarity = new double[trainingData.length];
    // Array to store the labels of the training data points
    int[] labels = new int[trainingData.length];
    // Calculate the similarity of the test feature with each training data point
    for (int i = 0; i < trainingData.length; i++) {
        similarity[i] = similarity(testFeature, trainingData[i]); // Compute similarity
        labels[i] = trainingLabels[i]; // Copy the label of the training data point
    }
    // Temporary variables for sorting the similarity scores and corresponding labels
    double temp = 0.0;
    int tempLabel = 0;

    // Bubble sort algorithm to sort the training data based on similarity in descending order
    for(int i = 0; i < trainingData.length - 1; i++) {
        boolean swapped = false;
        for(int j = 0; j < (trainingData.length - i - 1); j++) {
            // If the current similarity is less than the next similarity, swap them
            if(similarity[j] < similarity[j + 1]) {
                // Swap the similarity scores
                temp = similarity[j + 1];
                similarity[j + 1] = similarity[j];
                similarity[j] = temp;
                // Swap the corresponding labels to maintain relationship with their equivalent similarity scores
                tempLabel = labels[j + 1];
                labels[j + 1] = labels[j];
                labels[j] = tempLabel;
                swapped = true; // Set to true to indicate a swap occurred
            }
        }
        if(!swapped) {
            break;
        }
    }
    // Counter to count the number of "liked" labels (label value 1) in the top k most similar points
    int counter = 0;
    // Iterate over the top k most similar training data points
    for(int i = 0; i < k; i++) {
        if(labels[i] == 1) { // Check if the label indicates "liked"
            counter++;
        }
        // If more than half of the top k labels are 1, return 1 (indicating the movie is "liked")
        if(counter > k / 2.0) {
            return 1; // Majority voting results in "liked"
        }
    }
    return 0; // Return 0 if the majority of the top k labels are not 1
}
```

Figure six

We used a for loop in the main function to loop through the main code changing the K value each time and printed our accuracy for each odd value of K. We chose to use only odd values of K because in the event where the labels array up to the Kth index has an equal amount of likes and dislikes the algorithm would not be able to decide whether the movie is liked or not.This way we could determine which value of K yielded the highest accuracy. In model one it was when K = 11 with an accuracy of 70%.

```java
int mostAccurateK = 1; //finding most accurate k to use in final thing
double mostAccurateScore = 0.0; //finding most accurate score
for (int k = 1; k < 15; k+=2) {
    // Compute accuracy on the testing set
    int correctPredictions = 0;
    for (int i = 0; i < testingData.length; i++) {
        int prediction = knnClassify(trainingData, trainingLabels, testingData[i], k);
        if (prediction == testingLabels[i]) {
            correctPredictions++;
        }
    }

    // Add some lines here: ...

    double accuracy = (double) correctPredictions / testingData.length * 100;
    if (accuracy > mostAccurateScore) {
        mostAccurateScore = accuracy;
        mostAccurateK = k;
    }
    System.out.printf(k + " : %.2f%%\n", accuracy);
}
System.out.println("Most Accurate Prediction: " + mostAccurateScore + "| Most Accurate K: " + mostAccurateK); //to get the most accurate K
```

Figure seven

```
1 : 63.00%
3 : 63.00%
5 : 63.00%
7 : 63.00%
9 : 66.00%
11 : 70.00%
13 : 64.00%
Most Accurate Prediction: 70.0| Most Accurate K: 11

Process finished with exit code 0
```

Figure eight

3.2 Model Two:

In Model two, we introduced several key improvements to refine the KNN classifier and address the limitations of earlier iterations. These enhancements focused on normalisation and similarity computation.

One of the key enhancements was using the Z-score normalisation technique (Altman, 2013). Z-score normalisation reduces the influence of large-magnitude differences between features. Features like “Budget” and “Box Office” often have higher ranges compared to “IMDB” or

“Runtime”, which could lead to bias in the model if not normalised. Z-score normalisation ensures that all continuous variables contribute equally by centring the data and scaling it to a uniform range.

The Z-score normalisation is calculated by subtracting the mean (μ) of the feature from each individual value (X), then dividing the difference by the feature’s standard deviation (σ).

$$Z = \frac{X - \mu}{\sigma}$$

Equation one

This process standardised the dataset and prevented bias caused by feature magnitude differences(Singh and Singh, 2019).

Model one, despite achieving a higher accuracy of 70%, uses a value of K=11, which accounts for 11% of the entire training dataset. Utilising such a high percentage can be problematic as it fails to capture local patterns within the data(Zhang, 2021). Model two's use of K=3, with a lower accuracy of 67% and normalisation, provides a more balanced approach by incorporating a subset of nearest neighbours, which identifies local patterns.

3.3 Final model:

Classical Naïve Bayes and a model using Euclidean distance as a similarity metric were not included in the final selection as through experimentation and outside research we decided that using dot product was the better similarity metric (Maxim, Rodriguez and Wang, 2020). The final model we implemented is an ensemble that combines the first two models with a Gaussian Naïve Bayes model. By weighting the models according to their respective accuracy levels, we were able to achieve an overall accuracy of 85%. The implementation can be seen below in figure nine.

```java
public static void main(String[] args) {  Moadd-A *
    // Call the methods to generate predictions for each model
    int[] modelOneArray = ModelOne(); // Predictions from the first model
    int[] modelTwoArray = ModelTwo(); // Predictions from the second model
    int[] modelThreeArray = ModelThree(); // Predictions from the third model
    int[] testLabels = testingLabels(); // Labels from the test data

    // Define the weights for each model based on their accuracies
    // These weights will be used to determine the contribution of each model in the ensemble prediction
    double[] modelWeights = {0.5, 0.3, 0.4};


    int correctPredictions = 0;

    // Loop through each test to compare the predictions with the labels
    for (int i = 0; i < modelOneArray.length; i++) {

        // weightedEnsemblePredict method to determine predict based on test data label
        int predictedLabel = weightedEnsemblePredict(
                modelOneArray[i],
                modelTwoArray[i],
                modelThreeArray[i],
                testLabels[i],
                modelWeights
        );

        // Increment the counter if the ensemble's prediction matches label
        if (predictedLabel == testLabels[i]) {
            correctPredictions++;
        }
    }
    //print accuracy
    double accuracy = (double) correctPredictions / modelOneArray.length * 100;
    System.out.printf("Accuracy: %.2f%%\n", accuracy);
}
```

Figure nine

4. Naïve Bayes Classifiers:

Naïve Bayes, based on Bayes' Theorem, calculates probability given a set of features. Naïve Bayes is efficient as it is easily implemented with large datasets and can handle multiple data types such as strings and integers as well as continuous float types when using a Gaussian implementation (Kumar et al., 2022).

The Classic Naïve Bayes model was built using techniques previously learned in class such as the Naïve Bayes Formula (Equation two) and implementing the same correlation values found for the features in the KNN model. We tested each feature independently and then together to find the combination with the highest accuracy, this ended up being "Runtime", "Box Office" and "Budget". Whenever "IMDB" or "Rotten Tomatoes" were taken into account this lowered the accuracy to 57%.

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

Equation two

Since ratings are based on a scale we considered that a Gaussian Bayes model may return a higher accuracy. Gaussian Bayes is made specifically to handle continuous numerical data and classify data without having to convert it to bins meaning less chances of data loss (Kumar et al., 2022). By creating a Java model based on the Gaussian Bayes formula (Equation three) we returned an accuracy of 67% in the implementation seen in figure ten. While this is an improvement upon the previous 57% it is not an improvement on the overall accuracy of classical Naïve Bayes (Bustamante, Garrido and Soto, 2006).

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{\left(x_i - \mu_y\right)^2}{2\sigma_y^2}\right)$$

Equation three

Through experimentation and external research we concluded that for the current application, the Gaussian Bayes did not have a huge impact however could prove useful for other applications where all features are taken into consideration.

When compared to the previous KNN model, neither of the Bayes models achieved as high of an accuracy. This indicates that overall, KNN has proven to be more effective (Jadhav and Channe, 2016) for providing accurate predictions tailored to Alex's preferences.

```java
static int gaussianNaiveBayesClassify(double[][] trainingData, int[] trainingLabels, double[] testFeature) {
    double logPriorLike = Math.log(0.61); // log the prior probability of liking
    double logPriorDislike = Math.log(0.39); // log the prior probability of disliking

    // Make variables for the above
    double logProbLike = logPriorLike;
    double logProbDislike = logPriorDislike;

    // Add genre likelihoods
    logProbLike += genreLikelihood(trainingData, trainingLabels, classLabel: 1, testFeature);
    logProbDislike += genreLikelihood(trainingData, trainingLabels, classLabel: 0, testFeature);

    // For each numerical feature
    for (int feature = 8; feature < 14; feature++) {
        // mean and standard deviation for like and dislike
        double meanLike = calculateMean(trainingData, trainingLabels, classLabel: 1, feature);
        double stdDevLike = calculateStdDev(trainingData, trainingLabels, classLabel: 1, feature, meanLike);

        double meanDislike = calculateMean(trainingData, trainingLabels, classLabel: 0, feature);
        double stdDevDislike = calculateStdDev(trainingData, trainingLabels, classLabel: 0, feature, meanDislike);

        double weight = getFeatureWeight(feature);
        // Update the log-probabilities using the weight metric
        logProbLike += weight * gaussianLikelihood(testFeature[feature], meanLike, stdDevLike);
        logProbDislike += weight * gaussianLikelihood(testFeature[feature], meanDislike, stdDevDislike);
    }
        // compare the log probabilities and return the predicted case. if like has higher probability return 1 and otherwise 0
    return logProbLike > logProbDislike ? 1 : 0;
}
```

Figure ten

5. Conclusion:

The enhancements of Alex's movie prediction model improved its accuracy from 50% to a final accuracy of 85%. Key issues in the initial model, such as unbalanced genre weights and reliance on a single feature, were addressed by implementing techniques like Z-Score normalisation, one-hot encoding, and feature weighting. By implementing these techniques, balanced contributions from all features were achieved, resulting in a more robust model.

All enhancements were implemented using the permitted libraries and developed from first principles (Talin, 2023), reflecting a rigorous approach. The results demonstrate the importance of preprocessing and thoughtful feature selection in improving predictive accuracy while providing meaningful insights into Alex's movie preferences.

6. References:

Altman, E.I. (2013). PREDICTING FINANCIAL DISTRESS OF COMPANIES: REVISITING THE Z-SCORE AND ZETA ® MODELS. *RePEc: Research Papers in Economics*, Chapter 17, pp.428–456. doi:https://doi.org/10.4337/9780857936080.00027.

Bustamante, C., Garrido, L. and Soto, R. (2006). Comparing Fuzzy Naive Bayes and Gaussian Naive Bayes for Decision Making in RoboCup 3D. *Lecture Notes in Computer Science*, pp.237–247. doi:https://doi.org/10.1007/11925231_23.

Igual, L., Santi Seguí, Jordi Vitrià, Eloi Puertas, Petia Radeva, Oriol Pujol, Escalera, S., Francesc Dantí, Lluís Garrido and Springer International Publishing Ag (2017). *Introduction to data science : a Python approach to concepts, techniques and applications*. Cham Springer.

Jadhav, S.D. and Channe, H.P. (2016). Comparative Study of K-NN, Naive Bayes and Decision Tree Classification Techniques. *International Journal of Science and Research (IJSR)*, 5(1), pp.1842–1845. doi:https://doi.org/10.21275/v5i1.nov153131.

Kumar, M., Akash Gurrala, Vasireddy Bindu Hasitha and Rajesh, V. (2022). Introduction to Naive Bayes and a Review on Its Subtypes with Applications. *Chapman and Hall/CRC eBooks*, pp.1–14. doi:https://doi.org/10.1201/9781003164265-1.

Maxim, L.G., Rodriguez, J.I. and Wang, B. (2020). Defect of Euclidean Distance Degree. *Advances in Applied Mathematics*, [online] 121, p.102101. doi:https://doi.org/10.1016/j.aam.2020.102101.

Seger, C. (2018). *An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing*. [online] DIVA. Available at: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1259073&dswid=-6692 [Accessed 5 Dec. 2024].

Singh, D. and Singh, B. (2019). Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, 97(105524), p.105524. doi:https://doi.org/10.1016/j.asoc.2019.105524.

Syriopoulos, P.K., Kalampalikis, N.G., Sotiris Kotsiantis and Vrahatis, M.N. (2023). kNN Classification: a review. *Annals of Mathematics and Artificial Intelligence*. [online] doi:https://doi.org/10.1007/s10472-023-09882-x.

Talin, B. (2023). *Understanding and Applying First Principles Thinking: A Comprehensive Guide*. [online] MoreThanDigital. Available at: https://morethandigital.info/en/understanding-and-applying-first-principles-thinking-a-comprehensive-guide/ [Accessed 5 Dec. 2024].

Udovicic, M., Bazdaric, K., Bilic-Zulle, L. and Petrovecki, M. (2007). What we need to know when calculating the coefficient of correlation? *Biochemia Medica*, 17(1), pp.10–15. doi:https://doi.org/10.11613/bm.2007.002.

Zhang, S. (2021). Challenges in KNN Classification. *IEEE Transactions on Knowledge and Data Engineering*, 34(10), pp.1–1. doi:https://doi.org/10.1109/tkde.2021.3049250.