CS439 Data Science Project

IMDb Score Prediction

Mohammed Alnadi (ma1322) - 172007414
Umar Khattak (uk50) - 177009705

Introduction

For our project, we decided to implement machine-learning and data mining to predict the IMDb score of a movie. As both of us are relatively new to Python and machine-learning, we weren't really sure how to move forward with this, and had to learn as we went along with this project.

Using the CSV file movie_metadata.csv, which contained over 5000 rows (movies) and 28 columns, we filtered out the most relevant columns which we believed would give us the most accurate prediction.

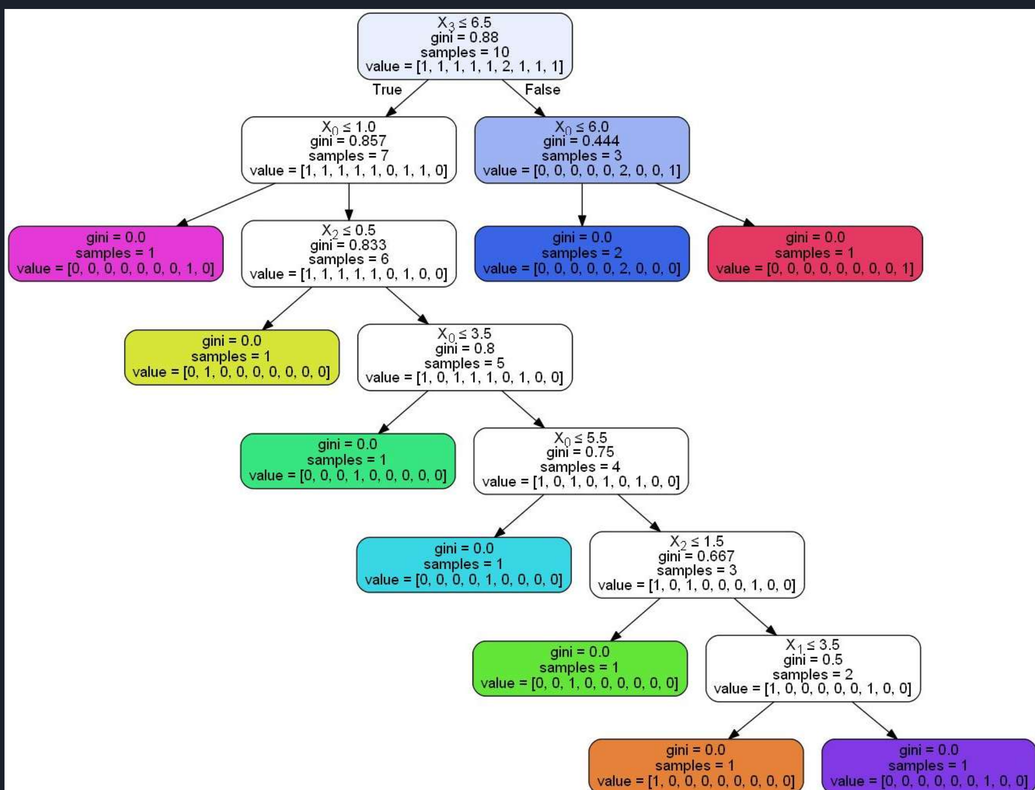
Introduction Part 2

In our first submission for the project, we concluded from our research that the higher the gross, the higher the IMDb score, and the same for movie facebook likes. (See Data Science Project.pdf)

So after creating a new dataframe with these specific columns which contained the movies gross, budget, total facebook likes, and IMDb score, we decided to use a Decision Tree to predict an output (y column which contained the actual IMDb score of the movie, hence why the column was used initially).

The new dataframe (X) contains 3891 rows and 5 columns (after deleting rows with any null values) which equaled out to 19,455 cells of data that the Decision Tree used. (The initial movie_metadata.csv dataset contained 141,204 cells of data)

At first to see how this will look like on a much smaller scale, we initially tried this out using only 10 rows (movies), below the Decision Tree that came out, along with records of our accuracy and prediction scores, where accuracy and precision were both 1.0 (See Data Science Final Project 4 (10 Movies) for Jupyter Notebook code)



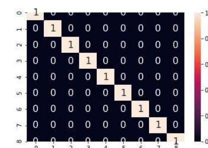

```
In [328]:
from sklearn.tree import DecisionTreeClassifier,export_graphviz
from sklearn.metrics import accuracy_score,recall_score,precision_score,confusion_matrix,f1_score
import matplotlib.pyplot as plt
import seaborn as sns

clf = DecisionTreeClassifier(max_depth=10).fit(X_train,Y_train)
print("Training:"+str(clf.score(X_train,Y_train)))
print("Test:"+str(clf.score(X_test,Y_test)))
pred = clf.predict(X_train)
confusion_matrix = confusion_matrix(y_true=Y_train,y_pred=pred)

sns.heatmap(confusion_matrix,annot=True,annot_kws={"size":16})
plt.show()

confusion_matrix
```

```
Training:1.0
Test:0.0
```




```
Out[328]: array([[1, 0, 0, 0, 0, 0, 0, 0, 0],
                 [0, 1, 0, 0, 0, 0, 0, 0, 0],
                 [0, 0, 1, 0, 0, 0, 0, 0, 0],
                 [0, 0, 0, 1, 0, 0, 0, 0, 0],
                 [0, 0, 0, 0, 1, 0, 0, 0, 0],
                 [0, 0, 0, 0, 0, 1, 0, 0, 0],
                 [0, 0, 0, 0, 0, 0, 1, 0, 0],
                 [0, 0, 0, 0, 0, 0, 0, 1, 0],
                 [0, 0, 0, 0, 0, 0, 0, 0, 1]], dtype=int64)
```

```
accuracy 1.0
f1 score micro 1.0
precision score 1.0
recall score 1.0
hamming_loss 0.0
classification_report              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         1
           2       1.00      1.00      1.00         1
           3       1.00      1.00      1.00         1
           4       1.00      1.00      1.00         1
           5       1.00      1.00      1.00         1
           6       1.00      1.00      1.00         1
           7       1.00      1.00      1.00         1
           8       1.00      1.00      1.00         1

    accuracy                           1.00         9
   macro avg       1.00      1.00      1.00         9
weighted avg       1.00      1.00      1.00         9

jaccard_similarity_score 1.0
zero_one_loss 0.0
```

Below is the Decision Tree of the entire dataset, as you can see its significantly larger than the tree for 10 movies, this is because we are using over 3800 movies, which means there are many more factors and numbers to take into consideration. (Took 20min to run :/)

```
In [159]:
print ("accuracy",metrics.accuracy_score(Y_train, pred))
print ("f1 score micro",metrics.f1_score(Y_train, pred, average='micro'))
print ("precision score",metrics.precision_score(Y_train, pred, average='macro'))
print ("recall score",metrics.recall_score(Y_train, pred, average='macro'))
print ("hamming_loss",metrics.hamming_loss(Y_train, pred))
print ("classification_report", metrics.classification_report(Y_train, pred))
print ("jaccard_similarity_score", metrics.jaccard_similarity_score(Y_train, pred))
print ("zero_one_loss", metrics.zero_one_loss(Y_train, pred))
```

```
accuracy 0.05969722936303913
f1 score micro 0.05969722936303913
precision score 0.014534967194253645
recall score 0.02216278211183815
hamming_loss 0.9403027706369609
classification_report              precision    recall  f1-score   support

           1       0.00      0.00      0.00         2
           2       0.00      0.00      0.00         1
           3       0.00      0.00      0.00         3
           4       0.00      0.00      0.00         1
           5       0.00      0.00      0.00         2
           6       0.00      0.00      0.00         2
           8       0.00      0.00      0.00         3
           9       0.00      0.00      0.00         5
          10       0.00      0.00      0.00         2
          11       0.00      0.00      0.00         2
          12       0.00      0.00      0.00         2
          13       0.00      0.00      0.00         1
          14       0.00      0.00      0.00        11
          15       0.00      0.00      0.00         3
          16       0.00      0.00      0.00         9
          17       0.00      0.00      0.00        10
          18       0.00      0.00      0.00         5
          19       0.00      0.00      0.00         8
          20       0.00      0.00      0.00         8
          21       0.00      0.00      0.00         7
          22       0.00      0.00      0.00        16
          23       0.00      0.00      0.00        12
          24       0.00      0.00      0.00        17
          25       0.00      0.00      0.00        18
          26       0.00      0.00      0.00        20
          27       0.00      0.00      0.00        25
          28       0.00      0.00      0.00        20
          29       0.00      0.00      0.00        30
          30       0.00      0.00      0.00        31
          31       0.00      0.00      0.00        26
          32       0.00      0.00      0.00        51
          33       0.00      0.00      0.00        41
          34       0.00      0.00      0.00        65
          35       0.00      0.00      0.00        64
          36       0.00      0.00      0.00        73
          37       0.00      0.00      0.00        85
          38       0.00      0.00      0.00        77
          39       0.00      0.00      0.00        90
          40       0.00      0.00      0.00       119
          41       0.07      0.24      0.11        94
          42       0.00      0.00      0.00       128
          43       0.67      0.02      0.03       126
          44       0.00      0.00      0.00       125
          45       0.07      0.34      0.11       140
          46       0.00      0.00      0.00       141
          47       0.00      0.00      0.00       145
          48       0.08      0.33      0.13       162
          49       0.04      0.53      0.08       140
          50       0.00      0.00      0.00       132
          51       0.00      0.00      0.00       132
          52       0.00      0.00      0.00       136
          53       0.00      0.00      0.00       138
          54       0.00      0.00      0.00       124
          55       0.00      0.00      0.00        91
          56       0.00      0.00      0.00        87
          57       0.00      0.00      0.00        94
          58       0.00      0.00      0.00        82
          59       0.11      0.12      0.11        78
          60       0.00      0.00      0.00        46
          61       0.00      0.00      0.00        50
          62       0.00      0.00      0.00        41
          63       0.00      0.00      0.00        22
          64       0.00      0.00      0.00        24
          65       0.00      0.00      0.00        13
          66       0.00      0.00      0.00        18
          67       0.00      0.00      0.00         6
          68       0.00      0.00      0.00         7
          69       0.00      0.00      0.00         5
          70       0.00      0.00      0.00         4
          71       0.00      0.00      0.00         2
          73       0.00      0.00      0.00         1

    accuracy                           0.06      3501
   macro avg       0.01      0.02      0.01      3501
weighted avg       0.04      0.06      0.02      3501

jaccard_similarity_score 0.05969722936303913
zero_one_loss 0.9403027706369609
```

Here are the prediction and accuracy scores for this Decision tree. Unfortunately, it was not what we expected, with accuracy being 0.06 and precision being 0.02. As we are both new to Python and machine-learning, we are not fully sure why this is the case. We assume this is the case because there are so many cases and data cells to take into consideration that we might have overfitted the data which ended up with a decreased accuracy, though we are not 100% sure.

Causes of overfitting include:

1. Overfitting due to the presence of noise. Mislabeled instances may contradict the class labels of other similar records.
2. Overfitting due to lack of representative instances. A lack of representative instances in the training data can prevent refinement of the learning algorithm.

A good model must not only fit the training data well but also accurately classify records it has never seen, and we assume this is something our tree lacked.

Conclusion

We can conclude based on the data we received that when it comes to predicting an IMDb score based on the data we have, a Decision Tree performs much more accurate and provides a higher accuracy and precision score when there is generally less data to take into consideration. In this case, the lesser amount of movies provided us with a higher accurate Tree.

Example being where our tree given 10 rows (movies) gave an accuracy of 1.0 whereas our tree with 3891 rows (movies) only gave an accuracy of 0.06.

Conclusion cont.

Is there a possibility to improve accuracy of a Decision Tree?

The answer is yes, according to an article from Analytics Vidhya, there are 8 methods to boost the accuracy of a Model

1. Adding more data
2. Treating missing and outlier values
3. Feature Engineering
4. Feature Selection
5. Multiple Algorithms
6. Algorithm Tuning
7. Ensemble Methods
8. Cross Validation

These are the factors we hope to look into and take into consideration for this project in the nearby future. Thank you for giving us this opportunity to get out our comfort zone and learn new things! We learnt a great deal from this project, and I hope you guys did as well. Have a great end of semester everyone!

Websites and Citations

Dataset from: https://www.kaggle.com/karrrimba/movie-metadatacsv

https://www.datacamp.com/community/tutorials/decision-tree-classification-python

https://www.kaggle.com/azzion/decision-tree-for-beginners

https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6

https://devpost.com/software/imdb-rating-prediction

https://medium.com/@artkulakov/visualizing-decision-trees-in-jupyter-notebook-with-python-and-graphviz-78703230a7b1

https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb

https://www.programcreek.com/python/example/89265/sklearn.metrics.precision_score

https://www.analyticsvidhya.com/blog/2015/12/improve-machine-learning-results/