We hereby declare that we are familiar with the rules regarding use of aids, academic integrity and reference technique for exams and other student works at Østfold University College.

We declare to be familiar with which actions are considered cheating/plagiarism in exams and other student works/tests.

We declare to be aware that violations of the rules on cheating may result in annulment and exclusion from Østfold University College, as well as exclusion from the right to give exams at all colleges and universities in Norway.

Classification and regression project

Google Play App Popularity Prediction

Briceno Benjamin, Cerutti Paolo, Delisle Thibault, Genin Victor,
Department of Computer Science,
Østfold University College

1. INTRODUCTION

The group, consisting of Briceno Benjamin, Cerutti Paolo, Delisle Thibault, and Genin Victor, are Computer Science students at Østfold University College. Our dataset comprises data from over 2.3 million Google Play Store apps, including attributes like name, category, price, download statistics, and ratings. These attributes enable predictions about post-release app performance, such as ratings and downloads[1]. Our project aims to address a key question: the advantage of a large dataset versus smarter algorithms. Specifically, we aim to develop a decision tree algorithm for predicting Google Play Store app downloads. Our primary objective involves comparing the performance of a basic decision tree algorithm (CART) on a large dataset with more advanced algorithms like XGBoost on a smaller dataset, all evaluated using a shared dataset with manipulated quantities for comprehensive performance testing.

2. ANALYSIS

2.1. RELATED WORKS

Many people have already tackled this problem in an attempt to predict an application's rating or popularity. The conclusions differ from one report to another: some feel that the data is not complete enough to make a viable prediction, while others find good results with relatively simple algorithms. So let's take a closer look at this problem, and come up with our own research and conclusion on predicting the number of downloads. [2] [3].

2.2. METHOD AND DESIGN

This dataset is structured as a Pandas DataFrame and possesses the following key characteristics:

- Number of Entries: The dataset comprises a total of 2,312,944 entries.
- Columns: It contains a grand total of 24 columns.
- Missing Data: Some rows have missing data.
- Number of possible values per column.
- Memory Usage: The dataset consumes approximately 361.8+ MB of memory.

The data cleansing process involved the following steps:

- Deletion of the "Installs," "Maximum Installs," "Developer Id," "Privacy Policy," and "Scraped Time" columns due to their limited value or the complexity of preprocessing required to make them useful.
- Deletion of rows containing missing values.
- Scaling of the "Rating count," "Price," "Size," "Last Update," and "Released dates" columns.
- Filtering out rows in the "Currency" column that contain "USD."
- Filtering rows in the "Minimum Android" column containing "Varies with device" and "and up."
- Creation of two new columns: "Released_Month" and "Released_Year."
- Creation of four new columns: “TLD,” “SLD,” “mailTLD,” “mailSLD.”
- Conversion of all remaining strings into integers.

2.3. TOOLS

We plan to use **Scikit Learn**, **CART**, **RandomForest**, **XGBoost** and **Classifium**. For preprocessing, we utilized **Knime**, a low-code software ideal for large datasets. Scikit-learn, a versatile Python library, is invaluable for our project. CART is a powerful machine learning algorithm for classification and regression tasks. RandomForest robustly combines multiple decision trees. XGBoost, known for accuracy and efficiency, sequentially trains decision trees.

In Scikit-learn's Random Forests, there are no built-in pruning options, but overfitting is mitigated by averaging predictions from multiple trees. XGBoost allows control over tree complexity through parameters like max_depth, min_child_weight, and gamma. It's specifically designed for boosting, with control over boosting parameters like learning_rate and n_estimators. Additionally, XGBoost provides options for setting feature importance scores based on feature usage across trees, allowing for feature selection or "winnowing" as needed.

Classifium excels in tabular data classification, minimizing overfitting and simplifying machine learning. While it lacks built-in feature engineering, external techniques can enhance results, especially when transitioning from XGBoost or RandomForest. Overall, for tabular data, Classifium offers superior accuracy and usability compared to neural networks.

2.4. APPROACH

In this section, we provide a comprehensive overview of our approach to model training, tailored to the specific algorithms employed.

2.4.1. CART

To establish a baseline, we initially executed the algorithm without any optimizations. Subsequently, to mitigate overfitting and enhance accuracy, we fine-tuned the model by adjusting the *maximum tree depth*. Once this was accomplished, we employed RandomizedSearchCV for hyperparameter tuning, aiming to identify the optimal values for parameters such as *minimum sample split*, *minimum samples leaf* and *criterion*.

2.4.2. RANDOM FOREST

When working with RandomForest, our strategy involved an iterative approach to identify the optimal model parameters in stages. We began by determining the best value for the *maximum tree depth*, followed by the *number of estimators*, and subsequently, we fine-tuned the *minimum samples for splitting and the minimum leaf nodes*. This sequential optimization process allowed us to progressively refine the model's performance. Unfortunately, as detailed in the challenges section, our limitation in this scenario was constrained by computer memory even though increasing it may not have resulted in significant improvements.

2.4.3. XGBOOST

In data pre-processing, we adjusted 'Minimum Installs' for XGBoost compatibility and defined our target variable. For model tuning, we started with a default XGBoost classifier, then employed Grid Search Cross-Validation to pinpoint optimal hyperparameters (*max depth*, *min child weight*, *gamma*, *subsample*, *colsample by tree*). We displayed the best hyperparameters and their accuracy scores. Finally, we split the data into training and testing sets, creating a final XGBoost classifier with the best hyperparameters and regularization techniques. We set a learning rate of 0.01 and 5000 estimators. Model performance was rigorously assessed using accuracy, confusion matrix, and classification report on the test dataset.

3. CHALLENGES

Our initial challenge revolved around the need to differentiate between valuable data and irrelevant information while also effectively cleansing the dataset. Even though we had access to a sizable and well-balanced dataset, we encountered difficulties in uncovering valuable insights for the model and in effectively expanding the dataset's attributes to provide the model with more meaningful data to operate on.
Then, while training with Random Forest, despite having access to 250GB of RAM, the algorithm exhibited such a voracious appetite for memory that it prematurely terminated due to memory exhaustion. In response to this issue, we opted to first determine the optimal hyperparameters of the model using a smaller yet still representative dataset. Subsequently, we intended to train the model on the larger dataset. When employing 66% of the dataset for hyperparameter tuning, we obtained the results that can be found in the next section.

4. RESULTS AND FINDINGS

4.1. CART

Without Hyperparameter Tuning:

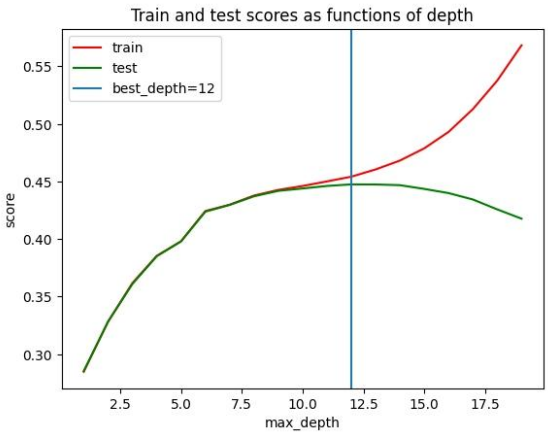
- Accuracy Training Score: 0.980
- Accuracy Testing Score: 0.339

The untuned CART model overfits the training data (high training score) but performs poorly on unseen data (low testing score).

With Max-Depth Tuning:

- Accuracy Training Score: 0.467
- Accuracy Testing Score: 0.448

Tuning the max-depth hyperparameter reduces overfitting, but the testing score remains relatively low.



RandomizedSearchCV Hyperparameter Tuning:

- Accuracy Training Score: 0.463
- Accuracy Testing Score: 0.450

Using RandomizedSearchCV, the best hyperparameters suggest a max depth of 12, entropy as the splitting criterion, and specific values for minimum samples per leaf and split. Training and testing scores are similar, indicating reduced overfitting. However, the testing score of 0.450 suggests room for further improvement.

In summary, RandomizedSearchCV improved generalization, but the model's overall performance may still be suboptimal. Additional tuning or exploring different algorithms may be needed for better results.

4.2. RANDOM FOREST

Accuracy:

- Train accuracy: 0.451
- Accuracy on Test Data: 0.460

This indicates that the model performed similarly on both the training and test datasets, suggesting that it may not be overfitting to the training data. Although it performed poorly in any case.

Classification Report:

The classification report details performance metrics for each class label, encompassing precision, recall, F1-score, and the number of instances (support) per class. Here's a succinct summary:

- The model excels in specific classes, like "10" and "1000," displaying high precision and recall.
- However, numerous classes exhibit low precision and recall, leading to diminished F1-scores.
- Classes such as "5," "50," "500," and others with very low or zero recall indicate challenges in accurate classification.

Classification Report:

	precision	recall	f1-score	support
0	0.69	0.01	0.02	1162
1	0.50	0.05	0.09	6889
5	1.00	0.00	0.00	8141
10	0.35	0.43	0.39	35665
50	1.00	0.00	0.00	21358
100	0.38	0.66	0.48	56861
500	1.00	0.00	0.00	24340
1000	0.46	0.70	0.55	51098
5000	1.00	0.00	0.00	18536
10000	0.48	0.67	0.56	33727
50000	1.00	0.00	0.00	10175
100000	0.41	0.61	0.49	15379
500000	1.00	0.00	0.00	3976
1000000	0.39	0.15	0.22	5154
5000000	1.00	0.00	0.00	1064
10000000	0.50	0.03	0.05	1020
50000000	1.00	0.00	0.00	132
100000000	1.00	0.00	0.00	78
500000000	1.00	0.00	0.00	6
1000000000	1.00	0.00	0.00	3
accuracy			0.42	294764
macro avg	0.76	0.17	0.14	294764
weighted avg	0.59	0.42	0.33	294764

These observations underscore the unsuitability of the scale created for this classification task, primarily due to the uneven class distribution. To address this, one potential solution could involve excluding certain classes, particularly the extremes, to better represent the overall class distribution.

Feature Importance:

The feature importances provides insights into the importance of each feature in making classification decisions. Features with higher importance values have a stronger influence on the model's predictions. In this case, it appears that "Rating Count" has the highest importance, followed by "Rating," while features like "Editors Choice" and "TotalLength" have relatively lower importance values.

Overall, the Random Forest Classifier shows mixed performance across different classes, with some classes being predicted well and others poorly. Further analysis and potentially model tuning may be needed to improve classification performance, especially for classes with low precision and recall.

Feature Importances:
Category: 0.0100
Rating: 0.3043
Rating Count: 0.4949
Free: 0.0062
Price: 0.0034
Size: 0.0081
Minimum Android: 0.0032
Content Rating: 0.0018
Ad Supported: 0.0300
In App Purchases: 0.0136
Editors Choice: 0.0002
Released_Month: 0.0017
Released_Year: 0.0181
Released: 0.0104
Updated: 0.0418
TLD: 0.0249
SLD: 0.0020
mailTLD: 0.0129
mailSLD: 0.0019
TotalLength: 0.0108

4.3. XGBOOST

Accuracy:

- Train accuracy: 0.565
- Accuracy on Test Data: 0.476

This suggests that the model is having difficulty correctly classifying data points. It is also notable that the model is overfitted to the training data.

Confusion Matrix:

The confusion matrix provides insights into the model's performance across different classes. In this case, it's apparent that the model struggles to make accurate predictions for most classes. The number of true positives (correctly predicted instances) is notably low compared to the total number of samples for each class. This indicates that the model encounters difficulty distinguishing between different classes.

Confusion Matrix:

[[	88	383	6	579	0	97	0	8	0	1	0	0	0	0	0	0	0	0	0	0]
[	59	1189	85	4580	5	966	0	2	0	2	0	0	0	1	0	0	0	0	0	0]
[	11	495	77	5660	10	1883	0	5	0	0	0	0	0	0	0	0	0	0	0	0]
[	10	611	73	20082	102	14597	4	185	1	0	0	0	0	0	0	0	0	0	0	0]
[	2	116	15	7670	109	13040	13	392	0	1	0	0	0	0	0	0	0	0	0	0]
[	2	122	14	10955	123	39039	461	6117	12	16	0	0	0	0	0	0	0	0	0	0]
[	1	16	2	1974	16	12220	576	9443	29	62	0	1	0	0	0	0	0	0	0	0]
[	0	11	0	1301	9	9720	449	35014	1404	3179	6	5	0	0	0	0	0	0	0	0]
[	0	1	0	103	1	810	14	8967	1888	6678	34	40	0	0	0	0	0	0	0	0]
[	0	0	1	71	2	1009	3	3985	1285	24900	1011	1457	3	0	0	0	0	0	0	0]
[	0	1	0	10	1	223	1	209	21	5199	1414	3079	8	9	0	0	0	0	0	0]
[	0	0	0	7	0	166	1	257	29	2014	1067	10929	481	427	0	0	1	0	0	0]
[	0	0	0	1	0	15	0	35	6	84	18	2062	651	1102	0	2	0	0	0	0]
[	0	0	0	1	0	5	0	32	4	89	16	736	442	3588	147	92	0	2	0	0]
[	0	0	0	0	0	0	0	1	0	8	0	26	10	624	163	230	2	0	0	0]
[	0	0	0	0	0	0	0	1	0	4	1	20	4	202	128	631	18	11	0	0]
[	0	0	0	0	0	0	0	0	0	1	0	2	0	9	2	79	22	17	0	0]
[	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	39	24	13	0	0]
[	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	1	0	0	0]
[	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0]]

Classification Report:


```
Classification Report:
 {'0': {'precision': 0.5086705202312138, 'recall': 0.0757314974182444, 'f1-score': 0.1318352059925094, 'support': 1162.0},
 '1': {'precision': 0.4037351443123939, 'recall': 0.17259399041950937, 'f1-score': 0.24181411429733576, 'support': 6889.0},
 '2': {'precision': 0.28205128205128205, 'recall': 0.009458297506448839, 'f1-score': 0.018302828618968387, 'support': 8141.0},
 '3': {'precision': 0.3789485602143639, 'recall': 0.5630730407962989, 'f1-score': 0.45301661421852274, 'support': 35665.0},
 '4': {'precision': 0.28835978835978837, 'recall': 0.005103474108062553, 'f1-score': 0.010029444239970556, 'support': 21358.0},
 '5': {'precision': 0.41623840494722253, 'recall': 0.6865690016003939, 'f1-score': 0.5182707051396938, 'support': 56861.0},
 '6': {'precision': 0.37844940867279897, 'recall': 0.023664749383730484, 'f1-score': 0.04454411878431676, 'support': 24340.0},
 '7': {'precision': 0.5415680633535953, 'recall': 0.6852322987201065, 'f1-score': 0.6049882938376343, 'support': 51098.0},
 '8': {'precision': 0.4035050224406925, 'recall': 0.10185584807941303, 'f1-score': 0.16265345681671334, 'support': 18536.0},
 '9': {'precision': 0.5895165490790284, 'recall': 0.7382809025409909, 'f1-score': 0.6555650628578951, 'support': 33727.0},
 '10': {'precision': 0.3964115503223998, 'recall': 0.13896805896805897, 'f1-score': 0.20579246106825788, 'support': 10175.0},
 '11': {'precision': 0.5953262882666958, 'recall': 0.7106443852005982, 'f1-score': 0.6478940036162077, 'support': 15379.0},
 '12': {'precision': 0.4071294559099437, 'recall': 0.1637323943661972, 'f1-score': 0.233542600896861, 'support': 3976.0},
 '13': {'precision': 0.601710548381687, 'recall': 0.6961583236321304, 'f1-score': 0.6454978861203562, 'support': 5154.0},
 '14': {'precision': 0.3704545454545455, 'recall': 0.15319548872180452, 'f1-score': 0.2167553191489362, 'support': 1064.0},
 '15': {'precision': 0.5837187789084182, 'recall': 0.6186274509803922, 'f1-score': 0.6006663493574489, 'support': 1020.0},
 '16': {'precision': 0.3235294117647059, 'recall': 0.16666666666666666, 'f1-score': 0.22, 'support': 132.0},
 '17': {'precision': 0.3023255813953488, 'recall': 0.16666666666666666, 'f1-score': 0.21487603305785125, 'support': 78.0},
 '18': {'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 6.0}, '19': {'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 3.0},

 accuracy': 0.47622165529033395, 'macro avg': {'precision': 0.3885824452033062, 'recall': 0.29381112678878574, 'f1-score': 0.2913022249034739, 'support': 294764.0},
 weighted avg': {'precision': 0.4485677405072329, 'recall': 0.47622165529033395, 'f1-score': 0.41434517993266895, 'support': 294764.0}}
```


The model excels in specific classes ("3," "7," "9," and "11"), achieving balanced precision and recall, leading to high F1-scores. However, it faces challenges in classes ("1," "5," "12," and "13") with imbalanced precision and recall. Notably low precision, recall, and F1-scores are observed in classes ("2," "4," "6," "8," "10," "14," "15," and "16"), posing classification difficulties.

The overall accuracy of the model is approximately 47.62%, indicating moderate correctness across all classes. Particularly low recall is identified in classes ("0," "2," "4," "6," "8," "10," "14," "15," "16," "17," "18," and "19"), signifying challenges in effectively identifying instances within these classes.

In summary, the XGBoost model exhibits inadequate classification performance, struggling with accurate categorization, resulting in low accuracy and F1-scores. While the results suggest varying costs in classification, exploration of technologies like instance weighting or similar mechanisms was deferred due to project time constraints.

4.4. CLASSIFIUM

We tried the Classifium method with a sample of 10 000 lines. We chose a sample size large enough to be representative of our dataset, but small enough not to take too long to run. We therefore opted for a run lasting almost 9 hours, with 2-fold cross-validation repeated twice.

Here are the results:

Error rate = 56.98% using 2-fold cross validation repeated 2 times.

Confusion matrix

	Actual: 0	Actual: 1	Actual: 10	Actual: 100	Actual: 1000	Actual: 10000	Actual: 100000	Actual: 1000000	Actual: 10000000	Actual: 100000000	Actual: 5	Actual: 50	Actual: 500	Actual: 5000	Actual: 50000	Actual: 500000	Actual: 5000000	Actual: 50000000
Predicted: 0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Predicted: 1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0
Predicted: 10	25.5	147.4	498.4	343.1	41.0	2.2	0.0	0.0	0.0	0.0	159.2	188.8	74.2	1.1	0.1	0.0	0.0	0.0
Predicted: 100	12.5	86.6	696.9	1287.2	298.8	32.0	9.8	1.0	0.0	0.0	116.8	504.8	403.9	34.4	8.9	1.0	0.0	0.0
Predicted: 1000	1.0	0.0	14.6	295.2	1179.6	158.0	11.2	1.0	0.0	0.0	0.0	31.0	338.4	278.6	13.1	2.2	0.0	0.0
Predicted: 10000	0.0	0.0	0.0	1.8	199.8	856.8	102.0	4.1	0.0	0.0	0.0	0.0	8.6	296.2	192.9	8.8	0.0	0.0
Predicted: 100000	0.0	0.0	0.0	0.0	2.0	69.9	352.2	60.9	2.2	0.0	0.0	0.0	0.0	7.2	120.1	81.9	8.0	0.2
Predicted: 1000000	0.0	0.0	0.0	0.0	0.0	0.5	34.5	97.9	23.9	1.1	0.0	0.0	0.0	0.0	0.6	38.6	22.9	1.9
Predicted: 10000000	0.0	0.0	0.0	0.0	0.0	0.0	0.1	7.1	8.2	1.9	0.0	0.0	0.0	0.0	0.0	0.6	5.1	1.9
Predicted: 100000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Predicted: 5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Predicted: 50	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Predicted: 500	0.0	0.0	0.0	1.2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.2	0.0	0.0	0.0	0.0	0.0
Predicted: 5000	0.0	0.0	0.0	0.2	10.4	14.8	0.4	0.1	0.0	0.0	0.0	0.0	0.6	10.9	0.1	0.5	0.0	0.0
Predicted: 50000	0.0	0.0	0.0	0.0	0.5	9.9	10.1	0.0	0.0	0.0	0.0	0.0	0.0	0.5	9.0	0.0	0.0	0.0
Predicted: 500000	0.0	0.0	0.0	0.0	0.0	0.0	1.5	2.9	0.1	0.0	0.0	0.0	0.0	0.0	0.1	1.4	0.0	0.0
Predicted: 5000000	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Predicted: 50000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

First of all, we observe an error of 56.98%, which is very high: more than half of the predictions have been incorrectly categorized. This statistic immediately shows that classifium doesn't improve classification in our

case. Looking at the matrix in more detail, we see that there is no aberration on the predictions, but that they are too scattered around the true value. Solutions that could potentially solve this problem would be to improve the predictors by increasing their number or quality (which in our case seems complicated), or to reduce the scale, keeping only the rungs containing a “1” and the “0”.

5. CONCLUSION

The results highlight the challenge in predicting this problem, which may be due to technology limitations or inherent complexity. Sensitivity analysis reveals that dataset size doesn't guarantee better results; a larger dataset can introduce noise and irrelevant data, affecting even the best models.

All algorithms converge on a common finding: the provided data lacks relevance for predicting app installs, with "rating" and "rating count" being the only meaningful predictors. To improve scores, enhancing the dataset with additional predictive features like usage time, updates, advertising, comments, etc., from app websites or other datasets, is crucial. Alternatively, refining the pre-processing step is an option but may require extensive time and testing.

Overall, this project allowed us to explore various algorithms, seek improvements, and grasp the challenges. Working with a large dataset exposed us to real-world complexities and professional insights.

6. REFERENCES

1. Prakash, G. (2021). *Google Play Store Apps*. Kaggle. https://www.kaggle.com/datasets/gauthamp10/google-playstore-apps

2. Maredi, R. (2020). *Analysis of Google Play Store Data set and predict the popularity of an app on Google Play Store*. Research Gate. https://www.researchgate.net/profile/Noman-Javed-2/publication/333059580_Stock_Price_Forecast_Using_Recurrent_Neural_Network/links/5cda0b1d458515712ea94121/Stock-Price-Forecast-Using-Recurrent-Neural-Network.pdf

3. Suleman, M., Malik, A., & Hussain, S. S. (2019). *Google play store app ranking prediction using machine learning algorithm*. Research Gate. https://www.researchgate.net/profile/Noman-Javed-2/publication/333059580_Stock_Price_Forecast_Using_Recurrent_Neural_Network/links/5cda0b1d458515712ea94121/Stock-Price-Forecast-Using-Recurrent-Neural-Network.pdf#page=67

4. Sklearn. *Decision tree classifier*. sklearn.tree.DecisionTreeClassifier — scikit-learn 1.3.1 documentation

5. Sklearn. *Random forest classifier. sklearn.ensemble.RandomForestClassifier — scikit-learn 1.3.1 documentation*

6. dmcl. *XGBoost*. Python Package Introduction — xgboost 2.0.0 documentation

7. *Classifium*. Classifium