Training ML models to predict games' properties

By: Artyom Kurguzkin

Introduction

A set of machine learning models was trained to predict game properties using supplied dataset. Models' performance was compared, and the best predictions can be found at the end of the report. This document mainly discusses the steps that I took to make these predictions and my reasoning behind them.

Data exploration

Supplied dataset had 403 rows and 386 columns. The first column is the unique name of the game, the last 5 are the labels that I aimed to train models to predict. The rest of the columns were 'ludemes' – properties of corresponding games as they appear in ludii software. Most of the target labels were categorical variables with multiple classes, except for UCT, which was continuous.

Data cleaning

Before proceeding with modeling, I ascertained that none of the columns were assigned wrong datatype, such as string representation of numbers. I checked if any of the columns contained any significant proportion of NA values. I found that the column 'int_int' consisted mostly of NA values (96.5%), so I dropped it. I also found 5 rows that had NA values in them. I found mode values for corresponding columns and substituted NA values with them.

Next, I dropped the first column with unique names. Learning algorithms I chose to use were not designed for language analysis, and unique values would not help in explaining data variability.

To detect outliers, I used Isolation Forest (Lim, 2022) unsupervised learning algorithm. I piped findings of the model into principal component analysis (PCA) transformer to visualise them on 2d graph (figure 1). The outlier rows were selected using index mask and removed from dataframe.

Feature selection

Most of the feature set data was binary-encoded, with uninterpretable column names, so I couldn't discern which ones have any importance. To reduce the number of columns, I decided to concentrate on removing columns that had redundant or the least amount of information.

I noticed that some of the features were one-hot encoded. In one hot encoding, one of the columns values can always be perfectly predicted from values of its counterpart, so they are perfectly correlated. I have built a correlation matrix (figure2) of the features and removed those that have a perfectly correlated counterpart.

I then assessed variance of each column and removed those that were below 30% threshold. I assumed that columns with little variability would not be useful in discerning label classes.

Label columns were encoded as integers. Mapping was saved to discern future predictions. I did it because I faced issues with cross validation algorithm in sklearn, which preferred integer representation for multiclass labels.

In our task, we were specifically asked to avoid discerning patterns between labels themselves, so I trained models for each label independently. For each label, the following procedure was followed…

Preparing data for modelling and feature extraction

For multiclass labels, I used stratified splitting (25% testing sample for model evaluation). To do it, I first checked if there were any labels that had only one corresponding instance. Single-instance classes would not be possible to predict, so I removed them. I found some in the OriginYear and Region labels.

I scaled training and testing feature samples using Min-Max scaling. I used it because it was less likely to affect the binary features, and I did not notice normal distribution in any of the continuous columns (except UTC, but it is our target label).

At this point, I still had too many features to run hyperparameter search. Procedurally iterating through all feature combinations also proved to be too time consuming, so I resolved to reduce the number of features using principal component analysis. To keep maximum amount of information intact, I iterated through various numbers of possible principal components and calculated their variance ratio. Then I plotted the increase in number of principal components vs cumulative explained variance (figure 3). I then found the optimum number of components that retained 95% of original data variance. The resulting reduction in the number of features was enough to proceed with model training. The extracted features were mostly normally distributed (figure 3). Technically it was achieved by fitting a pca transformer to training data. Then the testing set was transformed using the same fitted transformer. **Important notice**, everything that was done with testing feature set, was done to predictions feature set as well.

Model training and selection

I chose to try and train 3 models for each label. For categorical data, they are: K-Nearest Neighbor (KNN) classifier in bagging ensemble (50 instances); random forest (50 instances); and logistic regression. For continuous variable label (UTC) I used: KNN regression in bagging ensemble (50 instances); random forest regression (50 instances); and linear regression.

Hyperparameter tuning

For each model, I used grid search to find optimum hyperparameters. For classification models I used f1 score as the benchmark, and for regression models it was r^2.

Each model had their own set of hyperparameters I was testing. For KNN it was the number of neighbors. For Random Forest, I ran grid search with a random tree applying multiple possible depths, impurity measures, and leaf sizes (figure 4). For linear and logistic regression models, I iterated through different types of regularization and applied a range of regularization weights (C value).

Grid search was run with cross validation in hopes to find the best possible model settings.

Once optimum hyperparameters were identified, I trained the model with chosen hyperparameters and calculated f1 or r2 score using the test set.

Once all 3 models were trained for a given label, I chose the one that received the highest score and made prediction for that label using the prediction set. All predictions are recorded in the table below. I recorded: the game name (to make it possible to identify the prediction instance), predicted label values, and models that made the prediction with their performance metrics.

Game name	**Category**	**Region**	**BestAgent**	**OriginYear**	**UCT**
Main Tapak Empat	Hunt	Southern Asia	Alpha-Beta	1935	52.366
Boxijn Barildaan	War	Northern Asia	Alpha-Beta	1963	52.114
Provincial	Race	Northern Europe	Alpha-Beta	1300	60.814167
Tsoro Additional Capture	Sow	Eastern Africa	Alpha-Beta	1964	59.508
Marelle Quadruple	Space	Eastern Asia	Alpha-Beta	1694	64.886
Model used & metric	Logistic regression (f1=0.9)	KNN with bagging (f1=0.72)	KNN with bagging (f1=0.72)	Random Forest (f1=0.12)	KNN with bagging (r2=0.22)

Discussion

The OriginYear label is very unlikely to be correctly predicted. The label had a significant imbalance among its classes, as most of the games were developed more recently.

Random Forest, overall, seems to perform badly with PCA-reduced data. I think it struggles to find meaningful rules for tree growing where other algorithms use distance between points to calculate their predictions.

KNN preferred almost always to keep k=1 (from grid search hyperparameter optimization), which suggests that there were very little or no groups identified in the dataset. It still performed relatively well most of the time by relying labels' class distribution (guessed the most frequent class).

It is very likely that the results could be better if I chose features manually instead of reducing them with PCA and variance threshold. The reduced features likely lost non-linear relationships to their labels.

There are possibly better models specifically designed for binary features that I could have applied here.

Conclusion

Using the supplied dataset, 3 models for each label were trained. The best one among them was then used to make a prediction. The feature set was very big, and a lot of dubious attempts were made to decrease their number. It is likely the case that some of the steps caused information loss and worsened models' performance.

References

Lim, Y. (2022). *Unsupervised Outlier Detection with Isolation Forest*. Retrieved from Medium: https://medium.com/@limyenwee_19946/unsupervised-outlier-detection-with-isolation-forest-eab398c593b2

Appendix

Figures referenced from the report's text

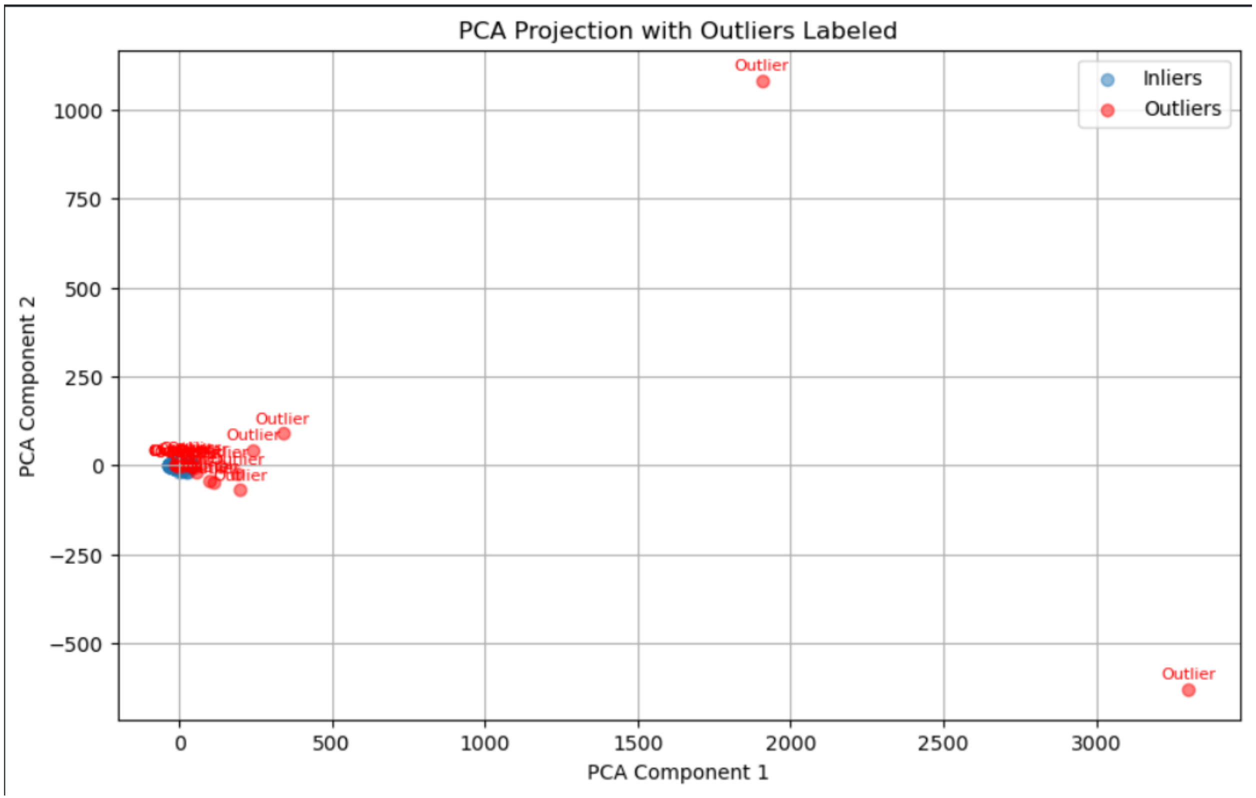


Figure 1: results of Isolation Forest model reduced to principal components.

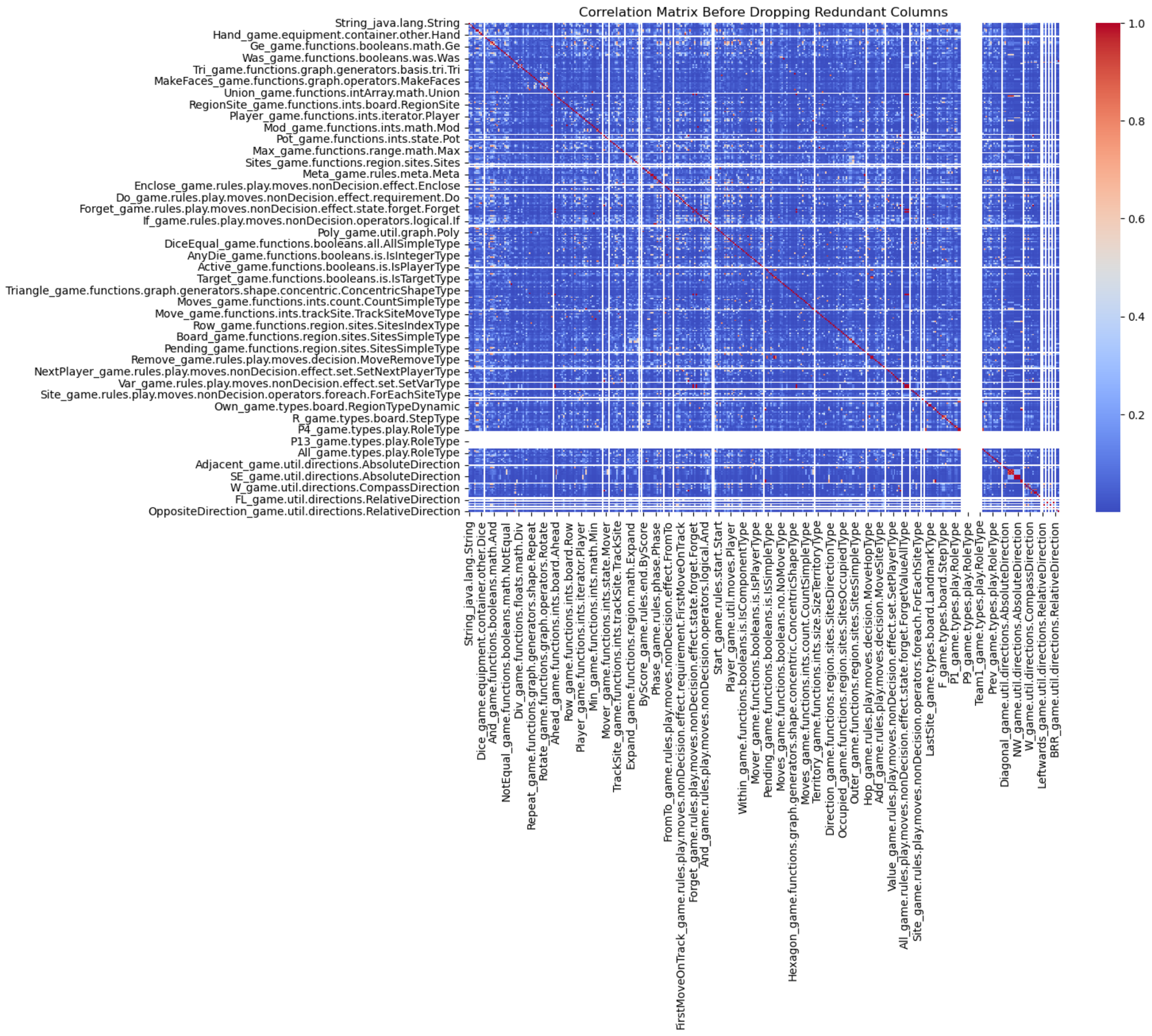


Figure 2: partial correlation matrix of the features used to drop one-hot encoded redundant collumns

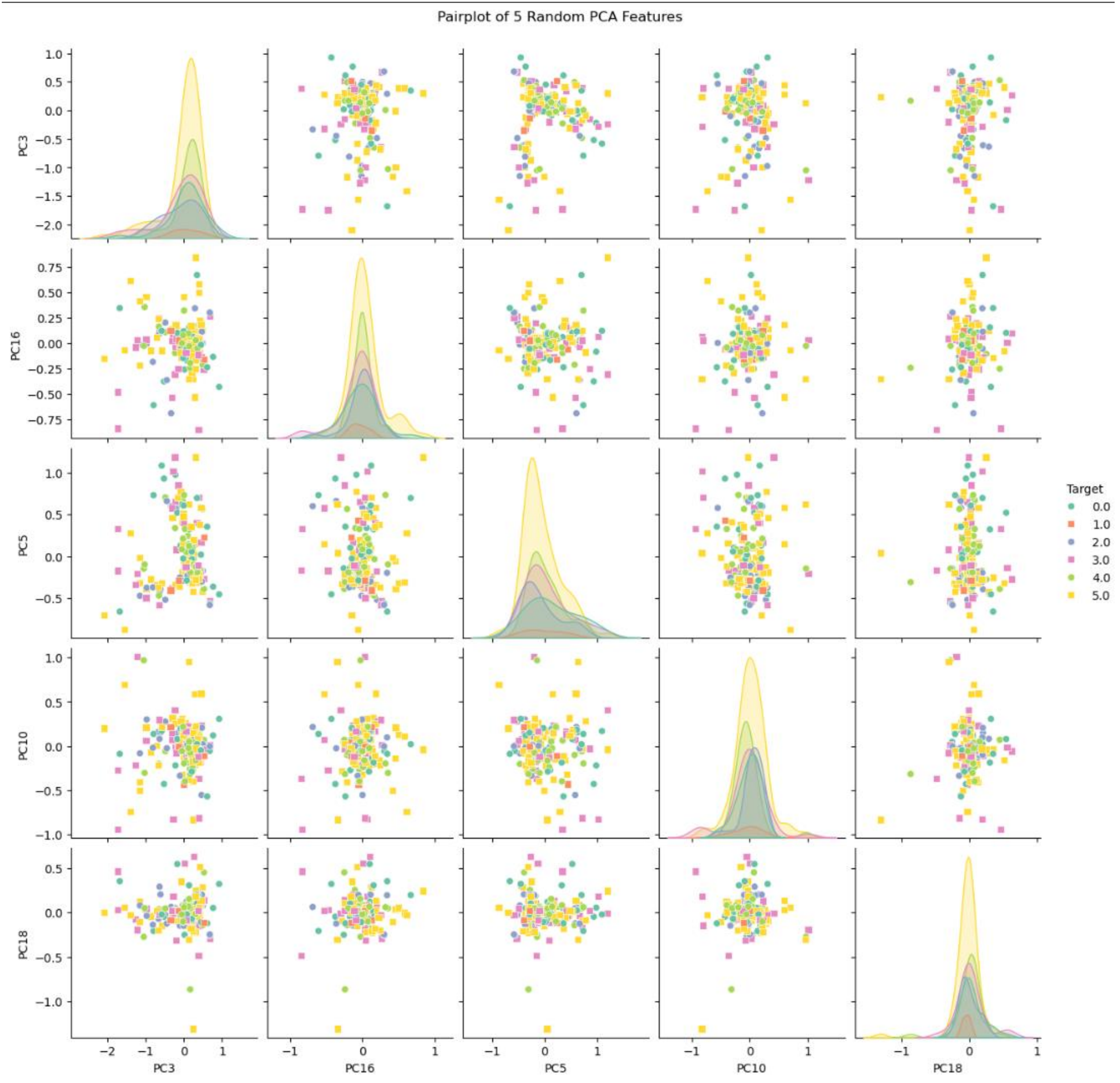
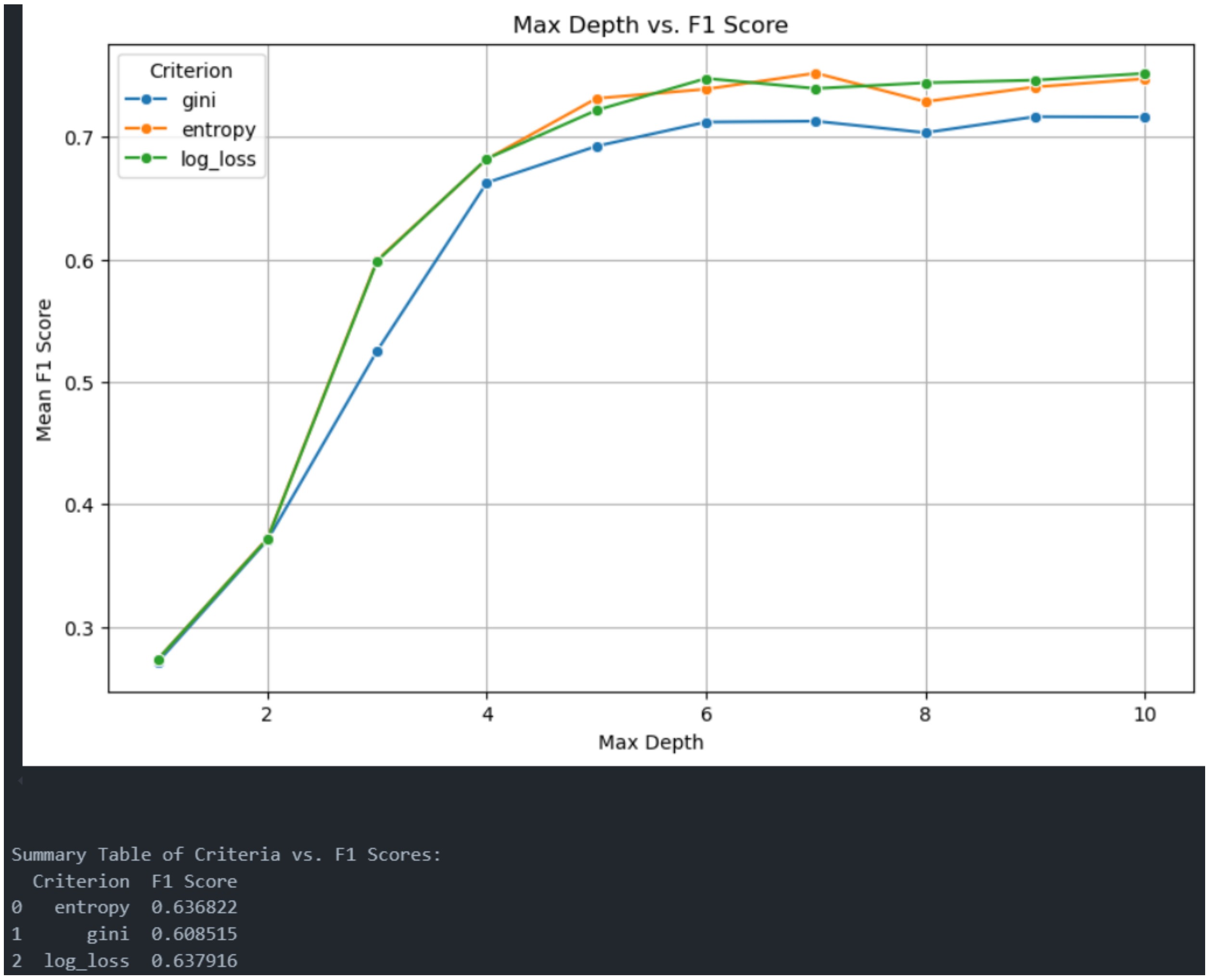


Figure 3: example of extracted principal components.



```
Summary Table of Criteria vs. F1 Scores:
  Criterion  F1 Score
0   entropy  0.636822
1      gini  0.608515
2  log_loss  0.637916
```

Figure 4: example of looking of optimum hyperparameters for predicting model. Decision tree in this case, which will be applied in random forest in the next step.