Decision Tree Implementation

– Prediction on League of Legends Game Results

Abstract

The aim of the practice is to predict the game results in League of Legends，based on the performances of the participating teams in the first ten minutes of the game.

After testing and tuning parameters on data processing and decision tree training, the best accuracy score reaches 0.729, with other scores at:
Precision: 0.7280534351145038, Recall: 0.7532082922013821, f1: 0.7404172731683649.

1 Introduction

League of Legends is a multi-player confrontation game developed and published by Riot Games. In the game, two teams(blue and red) of five players battle in player-versus-player combat, each team occupying and defending their half of the map.

Data provided for this practice is from Kaggle. Each team has 19 features, including kills, deaths, gold, experiences, champion level and so on. One round of the game usually lasts for 30-40 minutes. However, performances in the first ten minutes have large influences over the final result.

After analyzing and processing the given data, ID3 decision tree is used for training models.

2 Decision Tree and Programming Modules

A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical tree structure, which consists of a root node, branches, internal nodes and leaf nodes. In this practice, we experimented on both Information Gain and Gini impurity to identify the ideal features to split on.

Python is used for this implementation task. Pandas and Numpy are frequently used for Machine Learning and large data processing. Scikit is used for splitting data sets and validating prediction results. For test results visualization, seaborn and matplotlib are invoked.

3 Data Analyze

3.1 Data Cleanup

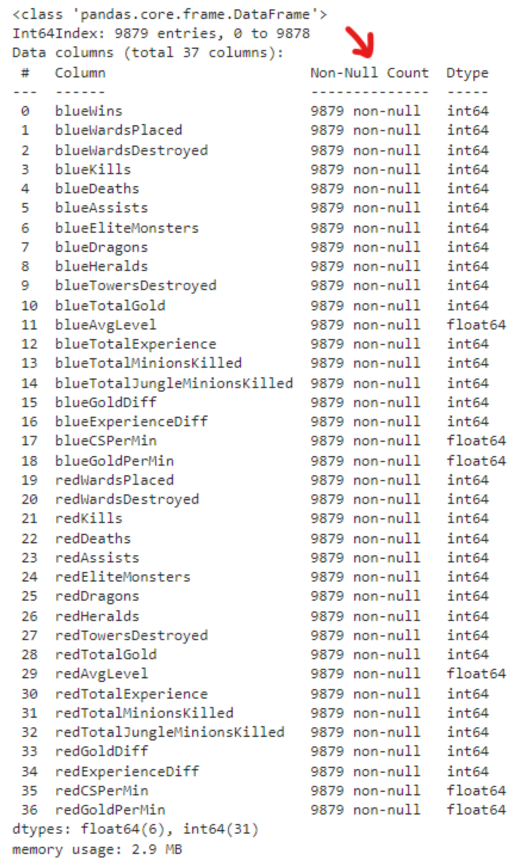
Data provided is clean enough. However, for practicing purposes, general steps[1] of validating and cleaning up are still processed.

The steps are:

a. Remove duplicates and constant features
b. Remove features with too much missing data
c. Remove samples with invalid data(based on the task)

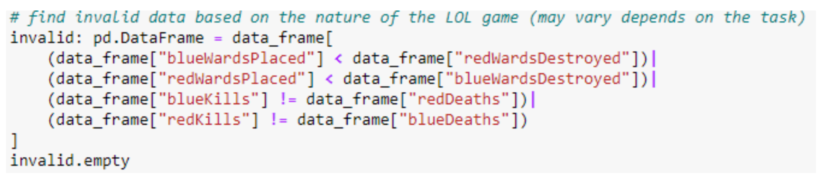
Using Pandas *drop_duplicates()* and *dataFrame.loc[:, (data_frame != data_frame.iloc[0]).any()]* can remove duplicates and constant features.

By calling *data_frame.info()*, we can see that there are no features with missing data. In general, if missing data exists, we need to drop the feature columns with too much missing data. In our example, no further modification needs to be applied.



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9879 entries, 0 to 9878
Data columns (total 37 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   blueWins                      9879 non-null   int64  
 1   blueWardsPlaced               9879 non-null   int64  
 2   blueWardsDestroyed            9879 non-null   int64  
 3   blueKills                     9879 non-null   int64  
 4   blueDeaths                    9879 non-null   int64  
 5   blueAssists                   9879 non-null   int64  
 6   blueEliteMonsters             9879 non-null   int64  
 7   blueDragons                   9879 non-null   int64  
 8   blueHeralds                   9879 non-null   int64  
 9   blueTowersDestroyed           9879 non-null   int64  
 10  blueTotalGold                 9879 non-null   int64  
 11  blueAvgLevel                  9879 non-null   float64
 12  blueTotalExperience           9879 non-null   int64  
 13  blueTotalMinionsKilled        9879 non-null   int64  
 14  blueTotalJungleMinionsKilled  9879 non-null   int64  
 15  blueGoldDiff                  9879 non-null   int64  
 16  blueExperienceDiff            9879 non-null   int64  
 17  blueCSPerMin                  9879 non-null   float64
 18  blueGoldPerMin                9879 non-null   float64
 19  redWardsPlaced                9879 non-null   int64  
 20  redWardsDestroyed             9879 non-null   int64  
 21  redKills                      9879 non-null   int64  
 22  redDeaths                     9879 non-null   int64  
 23  redAssists                    9879 non-null   int64  
 24  redEliteMonsters              9879 non-null   int64  
 25  redDragons                    9879 non-null   int64  
 26  redHeralds                    9879 non-null   int64  
 27  redTowersDestroyed            9879 non-null   int64  
 28  redTotalGold                  9879 non-null   int64  
 29  redAvgLevel                   9879 non-null   float64
 30  redTotalExperience            9879 non-null   int64  
 31  redTotalMinionsKilled         9879 non-null   int64  
 32  redTotalJungleMinionsKilled   9879 non-null   int64  
 33  redGoldDiff                   9879 non-null   int64  
 34  redExperienceDiff             9879 non-null   int64  
 35  redCSPerMin                   9879 non-null   float64
 36  redGoldPerMin                 9879 non-null   float64
dtypes: float64(6), int64(31)
memory usage: 2.9 MB
```

Based on the nature of the game, we can also validate if a sample's feature values are valid. For example, the kill amount from the blue team should be equal to the death amount for the red team.



```python
# find invalid data based on the nature of the LOL game (may vary depends on the task)
invalid: pd.DataFrame = data_frame[
    (data_frame["blueWardsPlaced"] < data_frame["redWardsDestroyed"])|
    (data_frame["redWardsPlaced"] < data_frame["blueWardsDestroyed"])|
    (data_frame["blueKills"] != data_frame["redDeaths"])|
    (data_frame["redKills"] != data_frame["blueDeaths"])
]
invalid.empty
```

The result shows that there is no invalid data to clean up.

3.2 Features Select

Decision tree classification is based on features. Constructing a data frame with features that are most relevant to the tasks can help with the learning results and training time spent.

Based on the nature of the game itself, we could remove: "gameId", "blueFirstBlood", "redFirstBlood" from all features, as they are not relevant enough. At this stage, we have in total 36 features, 18 for each team. Decision tree ID3 performs better when the feature amount is under 20. It is in need of reducing features to a reasonable amount.

By observing the data, we noticed that the features mostly come in pairs. Besides, since LOL is a confrontation game (also inspired by the code frame provided), the differences between red team and blue team can have more impact over the winning aspect.

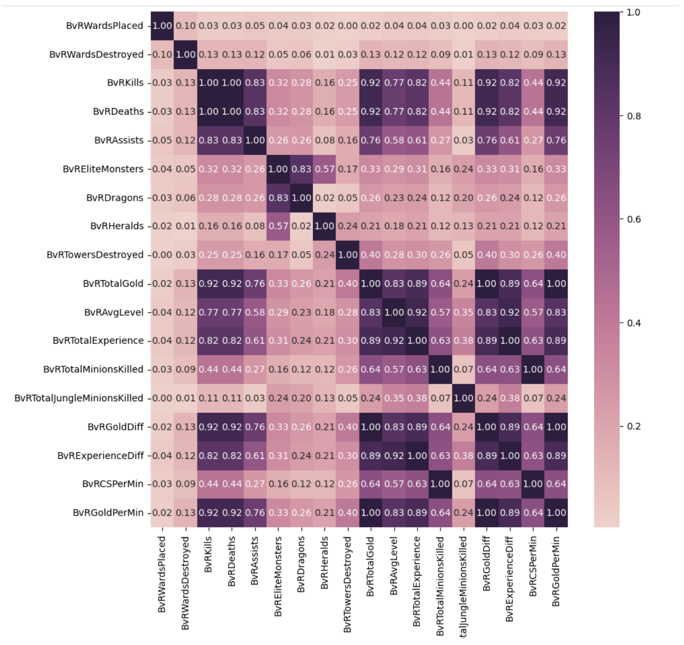
Thus, the following steps are executed:

Step 1: add difference data between two opposed features (blue-red) into the existing data frame

Same features for blue and red teams are substituted by the subtraction difference in between.

Step 2: remove highly-correlated features

Feature redundancy leads to large coefficient standard errors and shadows the true role of a feature in regression[2]. A way to determine redundancy is the correlation score. One thing that needs to be taken care of is that the correlation score can be positive or negative. When one variable increases as the other increases the correlation is positive; when one decreases as the other increases it is negative. In short, features with high absolute correlation scores should be reduced/removed[3].



```
correlated_features = get_correlated_features(feature_added_data_frame, correlated_threshold)
```

The threshold is tested and adjusted in a later stage. See for more details.

4 Discrete Data

If the feature values are continuous, it becomes difficult for the classifier(Decision Tree) to effectively draw this boundary and might have some skew in its results.[4] Here, qcut is used (similar to equal-frequency binning).

```python
def discrete_by_frequency(data_frame: pd.DataFrame, freq_threshold: int) -> pd.DataFrame:
    discrete_data_frame: pd.DataFrame = data_frame.copy()

    # if frequency is too low, quit discrete
    for column in data_frame.loc[:, data_frame.columns != LABEL_NAME]:
        if len(data_frame[column].unique()) < freq_threshold:
            continue
        discrete_data_frame[column] = pd.qcut(data_frame[column], freq_threshold, precision=0, labels=False, duplicates='drop')
    return discrete_data_frame

discrete_data_frame = discrete_by_frequency(data_frame, freq_threshold=3)
# best is 3 so far; see tests/validation section below
display(discrete_data_frame)
```

The frequency to discrete data is also experimented. See for more details.

5 Prepare Train Sets and Test Sets

Based on the provided code frame, we randomly take 20% of the overall data as the test set, and the rest as the train set. Python module "sklearn" provides a method "*train_test_split*" for this occasion. I/O for sklearn are usually numpy arrays, extraction or conversion between pandas dataframe and numpy array needs to be done.

6 Implementation of Decision Tree

Here, two ways of choosing the features to split on are implemented: Information Gain and Gini Impurity.

Both criteria are broadly similar and seek to determine which variable would split the data to lead to the underlying child nodes being most homogenous or pure [5].

a. Entropy-Information Gain:

$$E = -\sum_{i=1}^{n} p_i \, log_2(p_i) \qquad Information\,Gain = Entropy_{\,parent} - Entropy_{\,children}$$

The Entropy and Information Gain method focuses on purity and impurity in a node.

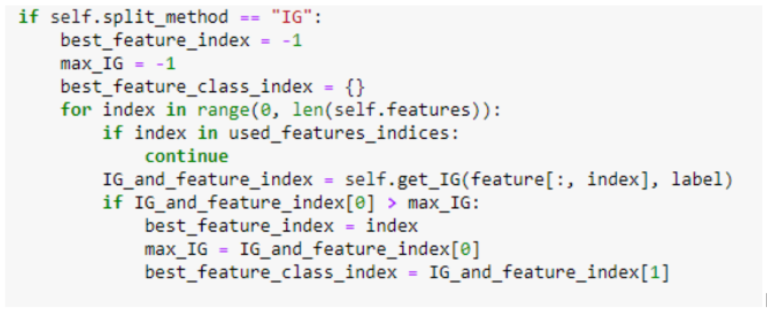
The higher the information gain is, the better the feature is to be chosen as the node to split on.

b. Gini Impurity:

$$Gini = 1 - \sum_{i=1}^{j} P(i)^2$$

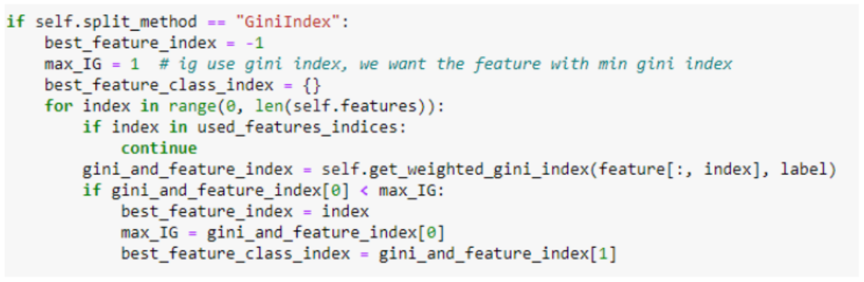
The Gini Index or Impurity measures the probability for a random instance being misclassified when chosen randomly. The overall Gini Index for a split is calculated similarly to the entropy as weighted average of the distribution across the sub-nodes[5].

Unlike Information Gain, we choose the feature with lower Gini Impurity as the best feature.



```
if self.split_method == "IG":
    best_feature_index = -1
    max_IG = -1
    best_feature_class_index = {}
    for index in range(0, len(self.features)):
        if index in used_features_indices:
            continue
        IG_and_feature_index = self.get_IG(feature[:, index], label)
        if IG_and_feature_index[0] > max_IG:
            best_feature_index = index
            max_IG = IG_and_feature_index[0]
            best_feature_class_index = IG_and_feature_index[1]
```

IG



```
if self.split_method == "GiniIndex":
    best_feature_index = -1
    max_IG = 1  # ig use gini index, we want the feature with min gini index
    best_feature_class_index = {}
    for index in range(0, len(self.features)):
        if index in used_features_indices:
            continue
        gini_and_feature_index = self.get_weighted_gini_index(feature[:, index], label)
        if gini_and_feature_index[0] < max_IG:
            best_feature_index = index
            max_IG = gini_and_feature_index[0]
            best_feature_class_index = gini_and_feature_index[1]
```

Gini

Depending on which impurity measurement is used, tree classification results can vary. Experiments are also run to evaluate which one is better for the data model we have.

7 Tuning Parameters

Here listed key parameters that can affect the learning results. K-folds is used for evaluating accuracy.

In Data Analyze and Processing:

(1) threshold for correlation score to remove redundant features

(2) frequency to discrete data

In Decision Tree Model:

(1) max depth of the tree

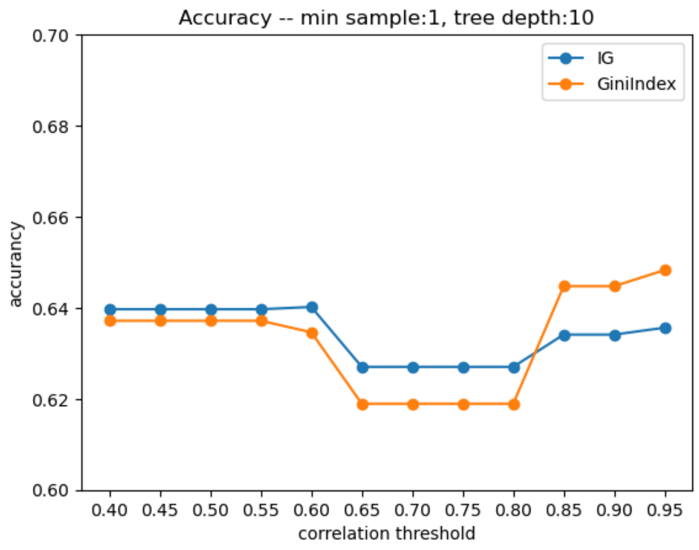
(2) threshold sample amount to stop splitting

(3) feature-select methods: Entropy-IG vs Gini

a. Threshold for Correlation Score to Remove Redundant Features

In general, loose correlation score thresholds perform better under a tree with maximum depth 10, and minimum split sample amount 1.

Before threshold gets below 0.85, splitting with Gini Index gives a better result, compared to Information Gain; After getting lower than 0.85, Information Gain produces higher prediction accuracy. Once the correlation threshold gets below 0.65, accuracy scores for both two splitting algorithms climb up again.



Thus, in the later testing for parameter adjusting, except changing only one parameter under two splitting algorithms, it's also essential to compare performances with different correlation thresholds.

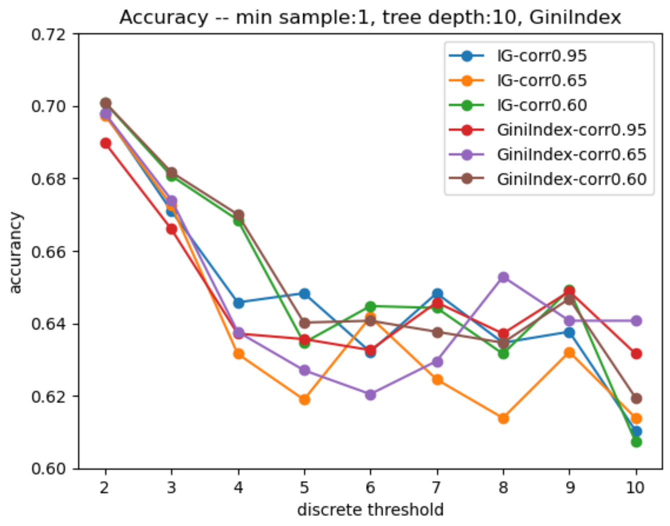
Based on the analysis above, threshold=0.85, 0.65, 0.6 will be chosen as test anchor points. As they represent three different trends for prediction accuracy.

b. Frequency for Discrete Data

Overall, there is a downtrend on accuracy scores whilst the frequency threshold for discretizing data increases.

Once the frequency threshold gets greater than 4, fluctuation happens on the prediction performances for both Gini Index and IG, which suggests low stability.

The second peak for accuracy scores happens when frequency threshold gets to 9(IG) and 8(Gini Index). The graph also suggests that the fluctuation with Gini Index is smoother than IG.



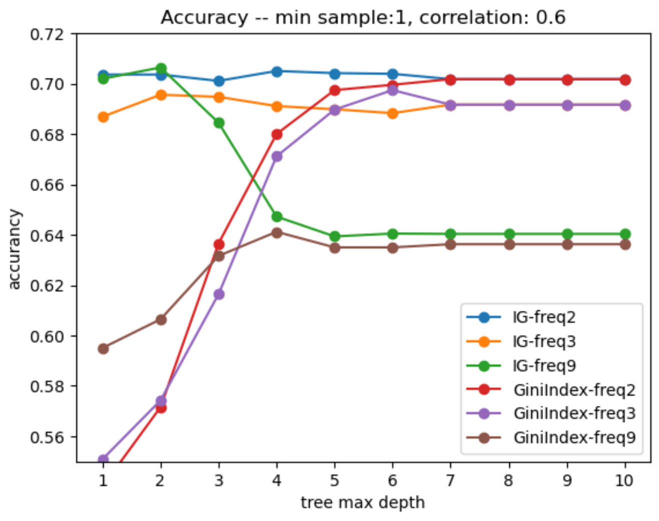
It is also an interesting fact that for both Gini index and IG, under the same frequency threshold, correlation threshold=0.6 performs better.

Before frequency threshold gets greater than 3, differences between performances for split algorithms under the same correlation score are not as obvious as when frequency threshold increases.

When discrete data frequency threshold equals to 3, and correlation threshold equals to 0.65 or 0.6, accuracy scores for IG and Gini Index are close.

When correlation is 0.95, IG and Gini Index predictions have obvious differences with the same set of data.

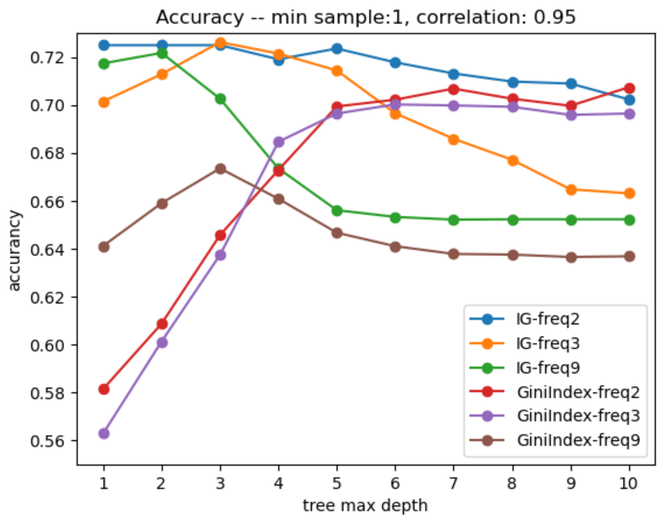
c. Max Depth for Decision Tree



By comparing two graphs, it came to notice that in this example, Information Gain performs generally better than Gini Index.

Under the same correlation threshold, if discrete data with a comparatively high frequency(9), accuracy is generally lower. This suggests that high discrete frequency should use small max depth for the tree, otherwise the training will overfit data.

For correlation=0.6, as the tree max depth reaches to 5, Gini and IG algorithms performance under the same discrete frequency gets closer to each other and overlaps after max tree depth to 7. The reason is we set a tight correlation threshold, and drop too many features. Thus, the tree layer gets empty as the depth gets deeper, due to there being no more features for splitting.



The accuracy peak for correlation=0.95 is higher than for correlation=0.65. So, we will choose correlation=0.95 for further testing.

For correlation=0.95,

Best accuracy for IG:

frequency=3 maxdepth=3 or frequency=2 maxdepth=5

Best accuracy for Gini Index:

frequency=2 maxdepth=10

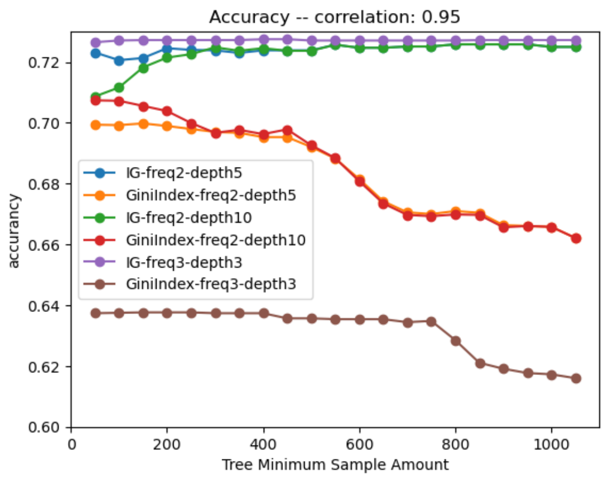
However, the result can also be affected by the minimum sample amount. The upper three will be tested under different minimum sample amounts for splitting.

d. Minimum Sample Amount to Stop Splitting

See from the graph, the best prediction accuracy happens when:

Correlation Threshold: 0.95, Discrete Frequency: 3, Tree Max Depth:3, Algorithm: Information Gain. Under this set of parameters, the minimum sample amount for splitting does not make a huge difference.

The minimum sample amount has a comparatively large impact over the Gini Index algorithm.



8 Conclusion

In general terms, under this task with the given data, Decision Tree with Entropy-IG generates more reliable predictions than with Gini Index.

The best accuracy score reaches 0.729, with other scores at: Precision: 0.7280534351145038, Recall: 0.7532082922013821, f1: 0.7404172731683649. The correlation threshold=0.95, discrete frequency=3, tree max depth=3 with Information Gain feature select produce this result.

Features in the given data have comparatively high correlation. For example, the amount of kills and jungle minion kills will cause direct/indirect changes in experiences, levels and gold amount. Afterwards, experiences and levels can affect the amount of kills again. Besides, in general terms, although the first 10 minutes performance can affect the final game result, it does not describe the whole round of the game comprehensively.

The way to tune correlation threshold for redundancy feature clearing can be better. In this practice, an iterative testing is used. It is also possible to do adaptive threshold based on the statistical analysis on features (etc.).

It is also essential to balance the data processing and learning model build. Take *Section7[c]* as an example, high discrete frequency works better with smaller max depths for the tree, otherwise the training will overfit data.

As mentioned above, since features in the given data have cross and iterative influences over each other, we might need to consider multiway trees.

10 References

[1] https://www.obviously.ai/post/data-cleaning-in-machine-learning

[2] https://towardsdatascience.com/feature-selection-why-how-explained-part-1-c2f638d24cdb in Section: How to Identify Feature Redundancy?

[3] The Correlation Coefficient: What It Is, What It Tells Investors
https://www.investopedia.com/terms/c/correlationcoefficient.asp

[4] Should we use discrete or continuous input for decision trees
https://datascience.stackexchange.com/questions/22771/should-we-use-discrete-or-continuous-input-for-decision-trees

[5]
https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c