6) Decision Trees

- A "decision stump" is just a decision tree with only one split, so like a single if-else statement. These trees are sometimes also referred to as "weak learners"

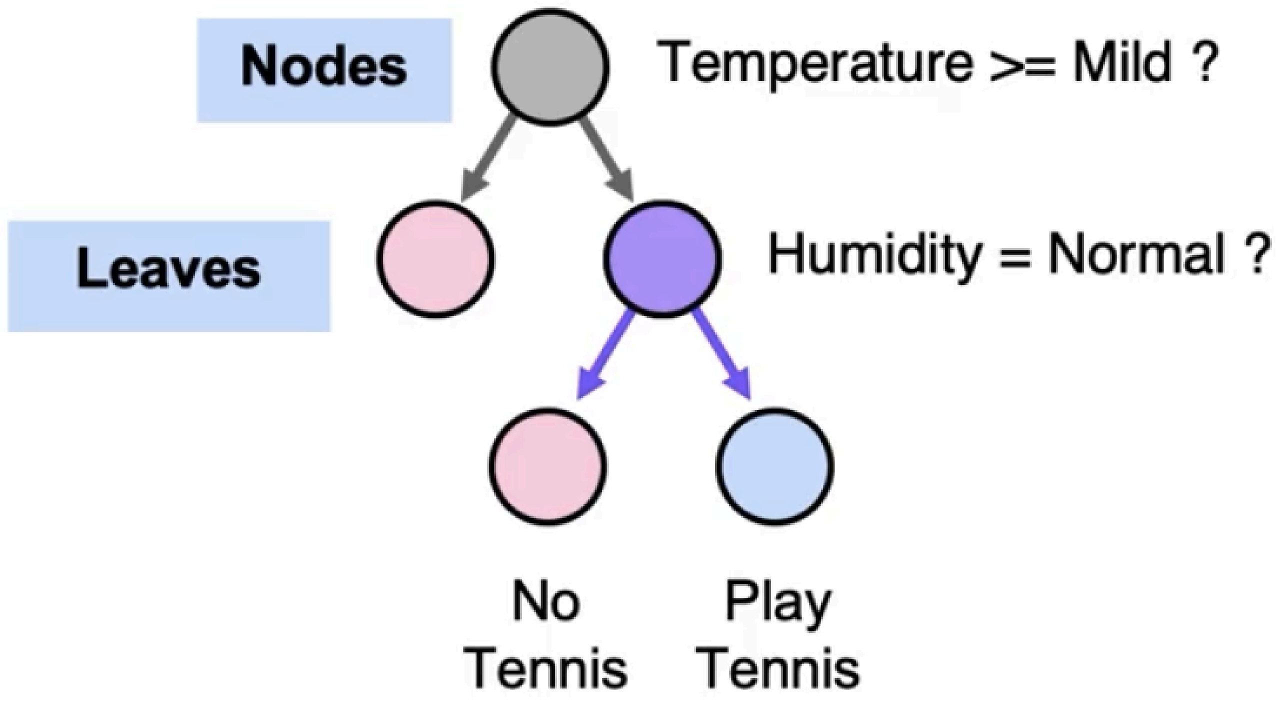
Pros:

- Easy to interpret and implement
- Able to handle any data category (binary, ordinal, and continuous)
- No preprocessing or scaling required
  Cons:
- Vulnerable to having high variance, tends to overfit. Variance can be reduced by "pruning" or by having a "max depth" that the tree can have, or the `max_features` hyperparameter. We can prune trees on the following basis:
  - classification error threshold
  - minimum amount of classes in a node

Hyperparameters

- `max_features`: It controls the maximum number of features that the decision tree algorithm considers when looking for the best split at each node in the tree. This hyperparameter can be useful for controlling the complexity of the tree and preventing overfitting. It can be set to the following values:
  - `None` **(default):** In this case, the algorithm considers all features at every split. This can lead to deep trees and may result in overfitting if you have a large number of features.
  - **Integer value:** You can specify an integer value `max_features=k`, where `k` is an integer less than the total number of features in your dataset. The algorithm will consider a random subset of `k` features at each split. This is known as "Random Feature Subsets" or "Random Subspace Method." It helps reduce overfitting and can speed up the training process. Setting `max_features` to a smaller value is often useful when you have a large number of features
  - **Float value:** You can specify a float value `max_features=f`, where `f` is in the range (0, 1). The algorithm will consider a fraction `f` of the total number of features at each split. For example, setting `max_features=0.5` means it will consider half of the features at each split
  - `auto` **or** `sqrt`**:** It is equivalent to setting `max_features` to the square root of the total number of features. It's a common choice for classification problems
  - `log2`**:** It is equivalent to setting `max_features` to the base-2 logarithm of the total number of features
  - `onethird`**:** It is equivalent to setting `max_features` to one-third of the total number of features

How it works



Outlook	Temperature	Play Tennis
Sunny	Hot	No
Sunny	Mild	No
Overcast	Hot	Yes
Rainy	Mild	Yes
Rainy	Cool	No
Overcast	Cool	Yes
Sunny	Mild	No
Rainy	Hot	No
Sunny	Cool	Yes
Overcast	Mild	Yes
Rainy	Mild	Yes
Overcast	Hot	Yes
Sunny	Hot	No

```
Outlook
├── Sunny
│   ├── Temperature
│       ├── Hot: No
│       ├── Mild: No
│       └── Cool: Yes
```

```
├── Overcast: Yes
└── Rainy
    ├── Temperature
        ├── Hot: No
        ├── Mild: Yes
        └── Cool: Yes
```

Algorithm

Remember, we want the possibility of homogenous (ie only one class per leaf) splits, but not actually result in homogenous splits cause else the model will overfit

Decisions trees will seek to split up the dataset into two datasets at every step, for which the decision is going to be easier, and then continue to iterate. We can keep iterating till:

- Leaf nodes are pure (ie only one class remains) (will overfit)
- Till a maximum depth is reached
- A performance metric (like accuracy) is achieved

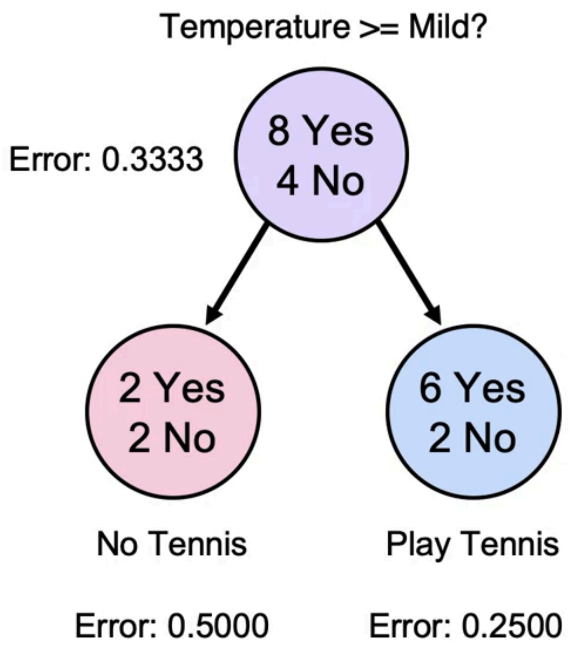
– Classification Error Equation

$$E(t) = 1 - \max_i [p(i|t)]$$

Splitting based on accuracy

To find the best split at a step. The best split is the split that maximises the information gained from the split

Information gained is defined by E(t) = 1 - max[p(i/t)]



– Classification Error Equation

$$E(t) = 1 - \max_i [p(i|t)]$$

– Classification Error Change

$$\begin{array}{r} 0.3333 \\ -\,{}^{4}\!/_{12} * 0.5000 \\ -\,{}^{8}\!/_{12} * 0.2500 \\ \hline = 0 \end{array}$$

Error of parent = 1-(8/12) = 0.3333

Error of left child = 1-(2/4) = 9.5

Error of right child = 1-(6/8) = 0.25

So in the above example, no information is gained because the classification error doesn't change. If this was the best split available, then we should stop splitting

Splitting based on entropy

$$H(t) = -\sum_{i=1}^{n} p(i|t) log_2[p(i|t)]$$

– Entropy Before

$$-\,{}^{8}\!/_{12}\, log_2({}^{8}\!/_{12}) \;-\; {}^{4}\!/_{12}\, log_2({}^{4}\!/_{12})$$

$$= 0.9183$$

– Entropy Left Side

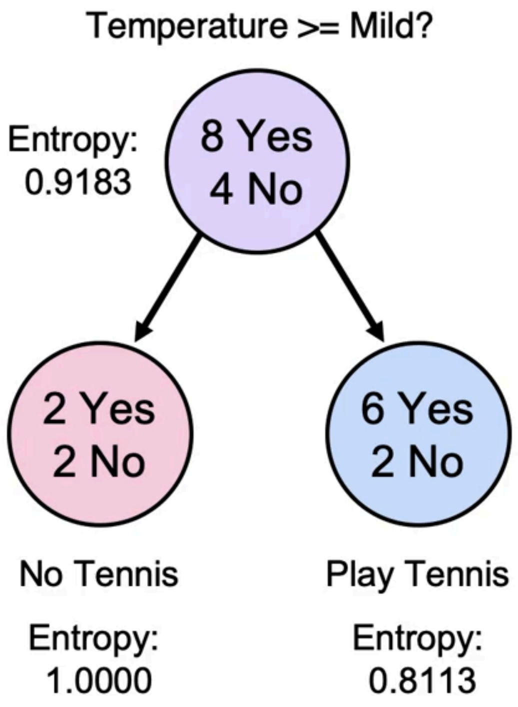
$$-\,{}^{2}/_{4}\, log_2({}^{2}/_{4}) \;-\, {}^{2}/_{4}\, log_2({}^{2}/_{4})$$

$$= 1.0000$$

– Entropy Right Side

$$-\,{}^{6}/_{8}\, log_2({}^{6}/_{8}) \;-\, {}^{2}/_{8}\, log_2({}^{2}/_{8})$$

$$= 0.8113$$



– Classification Error Equation

$$E(t) = 1 - \max_i [p(i|t)]$$

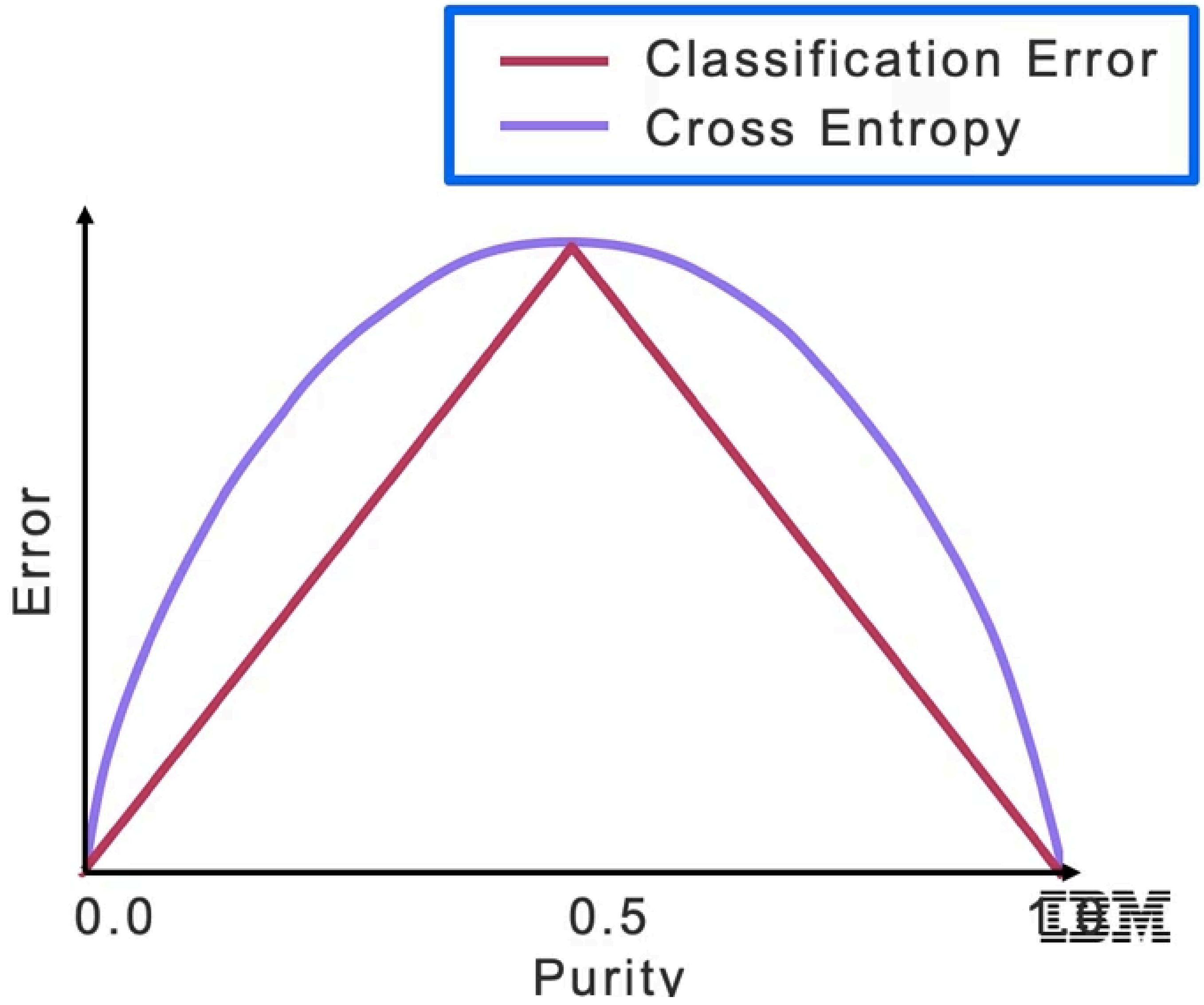
– Entropy Change

$$0.9183 - {}^{4}/_{12} * 1.0000 - {}^{8}/_{12} * 0.8113$$

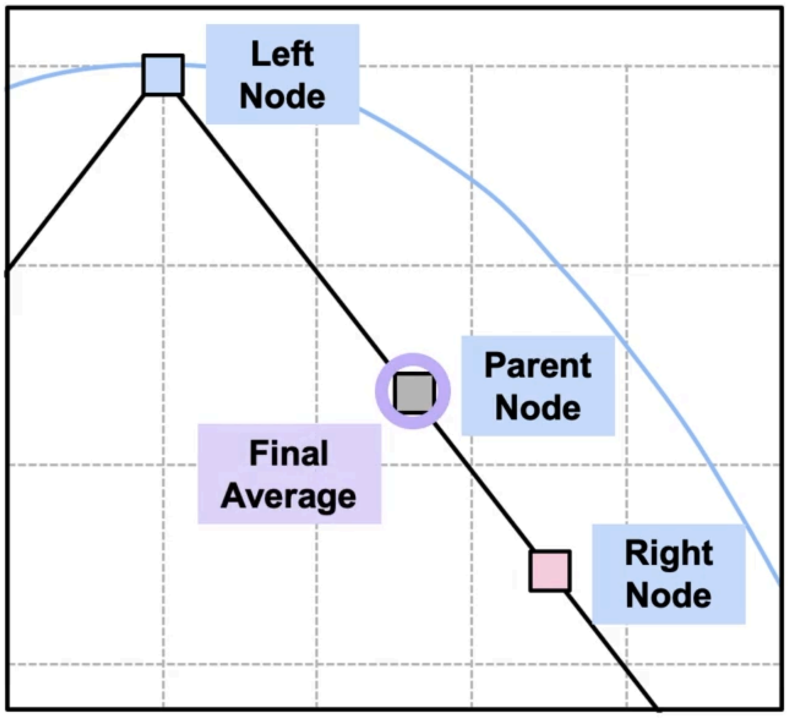
$$= 0.0441$$

Since the entropy has decreased by 0.0441, it means that information has been gained, and splitting can further occur

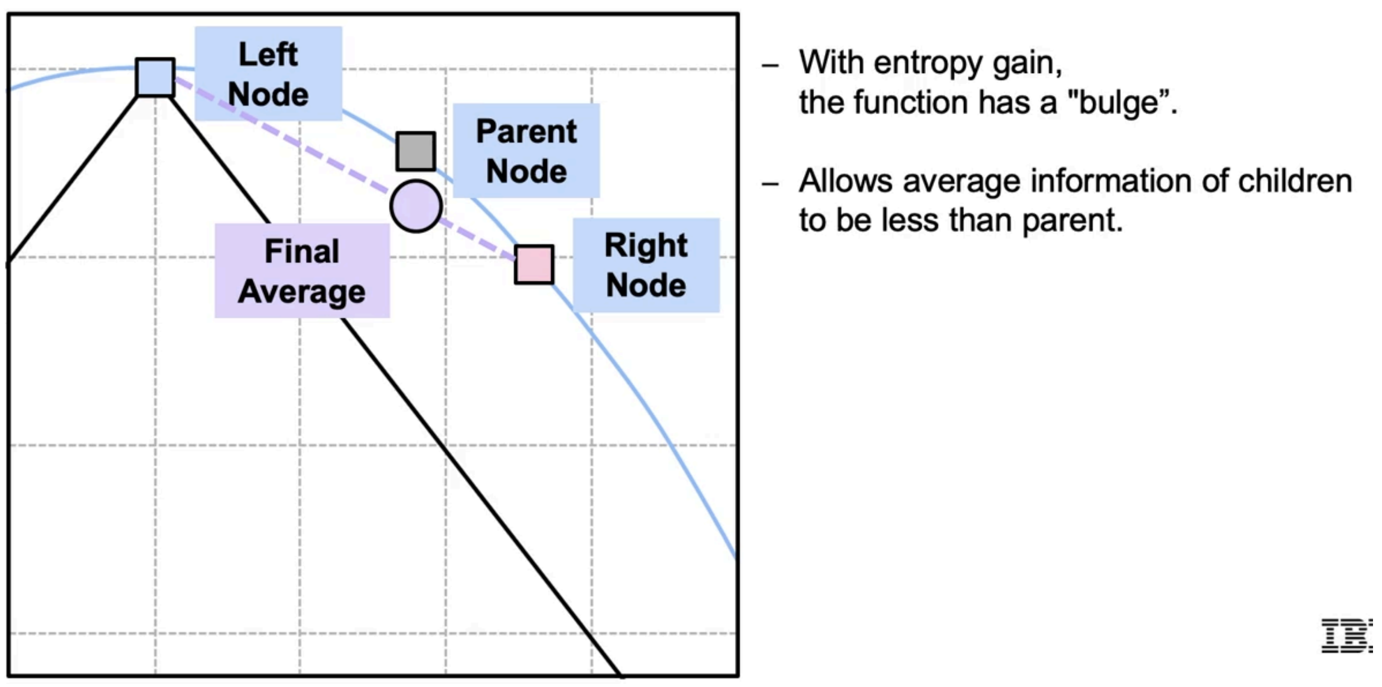
Classification error vs Entropy



Entropy can produce more information gain as



- With classification error, the function is flat.
- Final average classification error can be identical to parent.



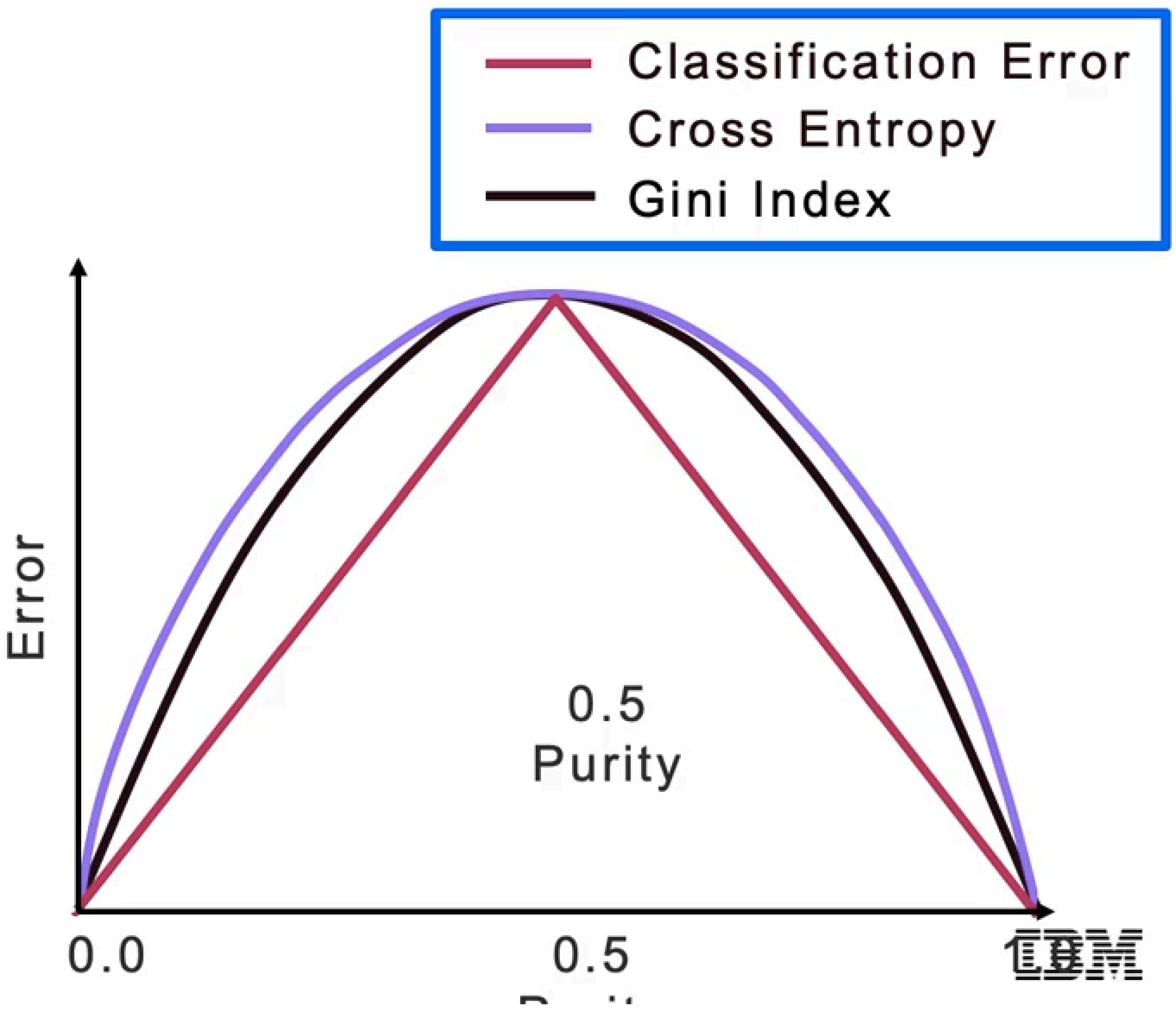
– With entropy gain,
the function has a "bulge".

– Allows average information of children to be less than parent.

In entropy, average of children will be lower than the parent

Gini index

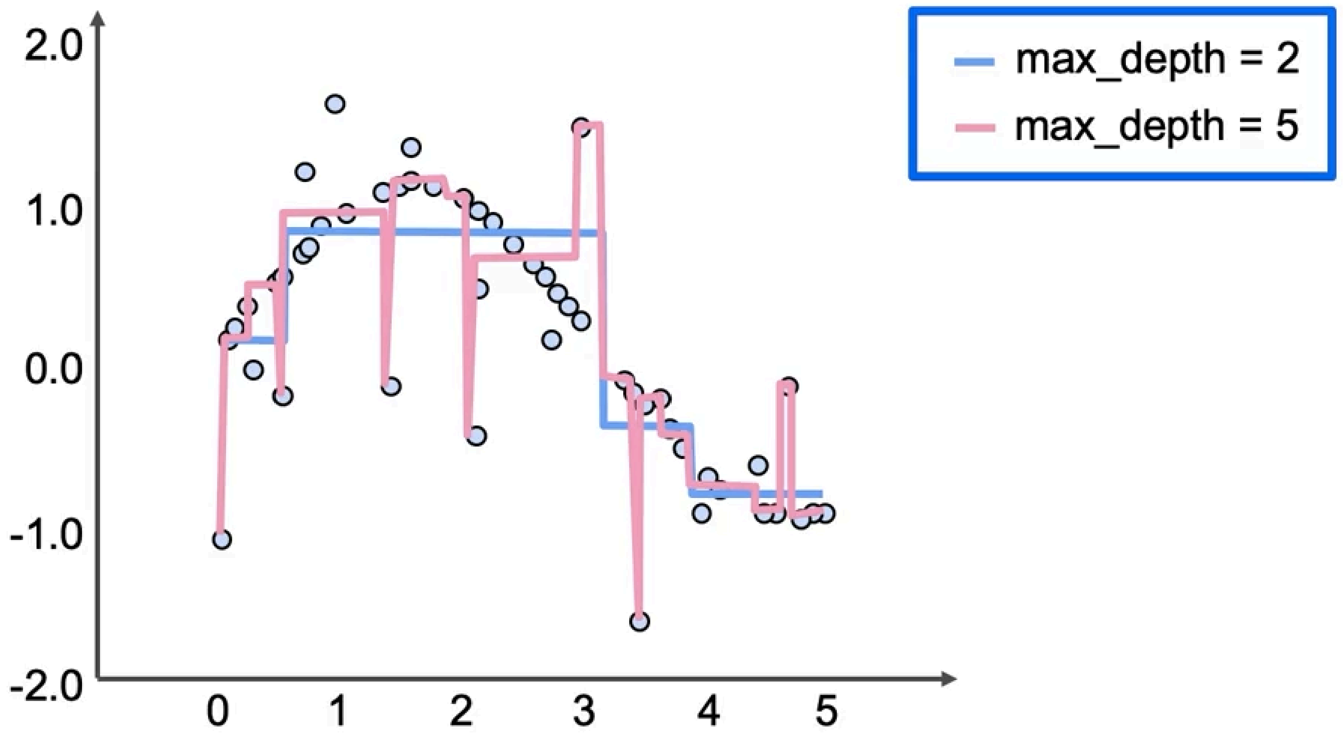
- Default metric used in `from sklearn.tree.DecisionTreeClassifier`
- Easier to compute since it doesn't contain logarithm



$$G(t) = 1 - \sum_{i=1}^{n} p(i|t)^2$$

Regression trees

We can use the same logic for regression. Here, the leaves will be averages of the members that satisfy the conditions. So our "regression line" for a 2D space can look something like



(Here, we can see that the model with depth=5 is overfit)