Data Mining → process of finding patterns and repetitions in large data sets

● C4.5 Algorithm

- Is used to generate a classifier in the form of a decision tree from a set of data that has already been classified
- Classifier here refers to a data mining tool that takes the data we need to classify and tries to predict the class of new data
- EXAMPLE:
  - Say we have a dataset that contains a bunch of patients[we know things like age, pulse, etc] → these are called attributes
  - Given these attributes, we want to predict whether the patient will get cancer[two classes: will get cancer or won't get cancer] → C4.5 is told the class for each patient
  - Using a set of patient attributes and the patient's corresponding class, C4.5 constructs a decision tree that can predict the class for new patients based on their attributes
    - Essentially, at each point in the flowchart, there is a question about the value of some attribute and depending on those values, he or she gets classified
- https://uh.edu/~smiertsc/4397cis/C4.5_Decision_Tree_Algorithm.pdf

● K-mean Algorithm

- A common clustering algorithm
- Works by creating a k number of groups from a set of objects based on the similarity between objects
- EXAMPLE:
  - Say we have a dataset of patients, in cluster analysis, these would be called observations
    - We know various things about each patient[age, pulse, blood pressure, etc] → these are called observations, a vector representing the patient
  - Now, how does one cluster together patients that have similar age, pulse, etc?
    - K-means picks points from the vectors to represent each of the k clusters → called centroids
    - Each patient will be closest to 1, what we have now are k clusters and each patient is part of a cluster
    - K-mean then finds the center for each of the clusters based on its cluster members
    - This center becomes the new centroid for each cluster, since the centroid is in a different place now, patients might now be closer to other centroids

- Repeat steps until the centroids no longer change

Apriori Algorithm

- Learns association rules and is applied to a database containing a large number of transactions
  - Association rule learning is a data mining technique for learning correlations and relations among variables in a database
- EXAMPLE:
  - Say we have a database full of supermarket transactions[each row is a customer transaction and every column represents a different grocery item]
  - By applying the algorithm, we can learn the grocery items that are purchased[aka association rules]
  - Result: you can find items that tend to be purchased together more frequently than other items[together are called itemsets]
- Setup: defining 3 things
  - Size of the itemset → do you want to see patterns for a 2 or 3 item set, etc?
  - Support → number of transactions containing the itemset divided by the total number of transactions
  - Confidence → the conditional probability of some item given you have certain other items in your itemset
- Algorithm
  - Join → scan the whole database for how frequent 1 itemsets are
  - Prune → those itemsets that satisfy the support and confidence move onto the next round for 2 itemsets
  - Repeat → this is repeated until we reach our previously defined size

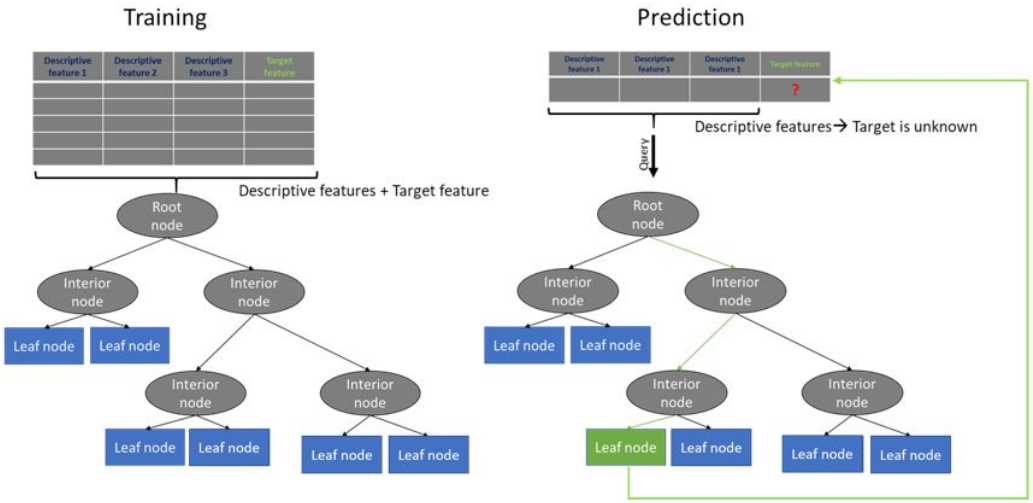
CART Algorithm

- Stands for classification and regression trees
- A decision tree learning technique that outputs either classification or regression trees
- A regression tree predicts a number or continuous value[like patients length of stay or price of a phone]

Implementation Example

- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.DecisionTreeRegressor
- Process of training a decision tree and predicting the target features of query instances is:
  - Present a dataset containing of a number of training instances characterized by a number of descriptive features and a target feature

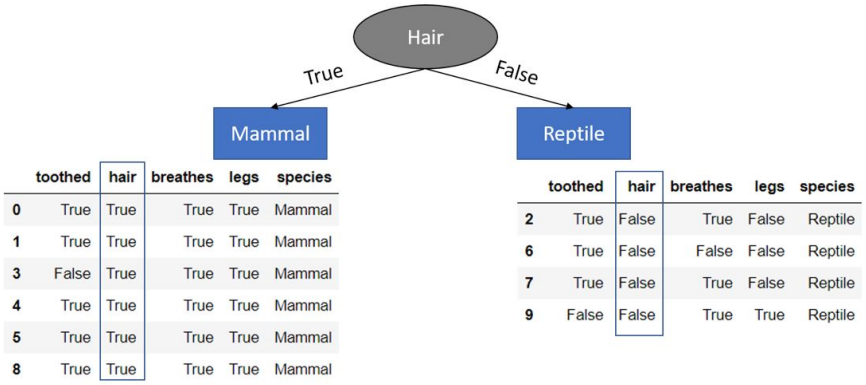
- Train the decision tree model by continuously splitting the target feature along the values of the descriptive features using a measure of information gain during the training process
- Grow the tree until we accomplish a stopping criteria → create leaf nodes which represent the predictions we want to make for new query instances
- Show query instances to the tree and run down the tree until we arrive at leaf nodes
- DONE



-

	toothed	hair	breathes	legs	species
0	True	True	True	True	Mammal
1	True	True	True	True	Mammal
2	True	False	True	False	Reptile
3	False	True	True	True	Mammal
4	True	True	True	True	Mammal
5	True	True	True	True	Mammal
6	True	False	False	False	Reptile
7	True	False	True	False	Reptile
8	True	True	True	True	Mammal
9	False	False	True	True	Reptile

-
  - Each leaf node should[in the best case] contain “mammals” or “reptiles”
  - Task is to find the best way to split the dataset such that this can be achieved
    - To split: we need to consider the dataset and look into how to split into two data sets of only mammals in data set 1 and only reptiles in data set 2
    - If hair → true[associated species is always a mammal]



	toothed	hair	breathes	legs	species
0	True	True	True	True	Mammal
1	True	True	True	True	Mammal
3	False	True	True	True	Mammal
4	True	True	True	True	Mammal
5	True	True	True	True	Mammal
8	True	True	True	True	Mammal

	toothed	hair	breathes	legs	species
2	True	False	True	False	Reptile
6	True	False	False	False	Reptile
7	True	False	True	False	Reptile
9	False	False	True	True	Reptile

-
- We split our dataset by asking the question if the animal has hair or not
    - This asking and splitting is key to decision tree models
- *What is the descriptive feature we should start with (root node) and in which order should we ask the questions(build the interior nodes), or use descriptive features to split the data set on?*
    - Would be useful to measure the “informativeness” of the features and use the features with the most “informativeness” as the feature which should be used to split the data on
    - Use the term **information gain** as a measure of “informativeness” of a feature
- **Math behind decision trees**
    - Entropy: used to measure the impurity of a dataset
        - Will be using this kind of informativeness measure in calculations
    - Also other types of measure which can be used to calculate the information gain
        - Gini index
        - Chi-square
        - Information gain ratio
        - Variance
    - Idea behind entropy[in simple terms]: imagine you have a lottery wheel which has 100 green balls. The set of balls within the lottery wheel can be said to be totally pure because there are only green balls[this set of balls has a entropy of 0(zero impurity)]
        - Consider a new lottery wheel in which 30 of these green balls are replaced by red and 20 by blue balls.
            - If you draw a ball from this wheel, the probability of receiving a green ball has dropped from 1.0 to 0.5
            - Since the impurity increased, the purity decreased, hence the entropy increased
            - **The more “impure” a dataset, the higher the entropy**
            - **The less “impure” a dataset, the lower the entropy**
    - Formal definition of Shannon’s entropy[serves as the baseline for the information gain calculation]:

- $$H(x) = -\sum_{for\ k\ \in target} (P(x = k) * log_2(P(x = k)))$$
- $P(x = k)$ is the probability that the target feature takes a specific value k

  Green balls: $H(x = green) = 0.5 * log_2(0.5) = -0.5$
  Blue balls: $H(x = blue) = 0.2 * log_2(0.2) = -0.464$
  Red balls: $H(x = red) = 0.3 * log_2(0.3) = -0.521$

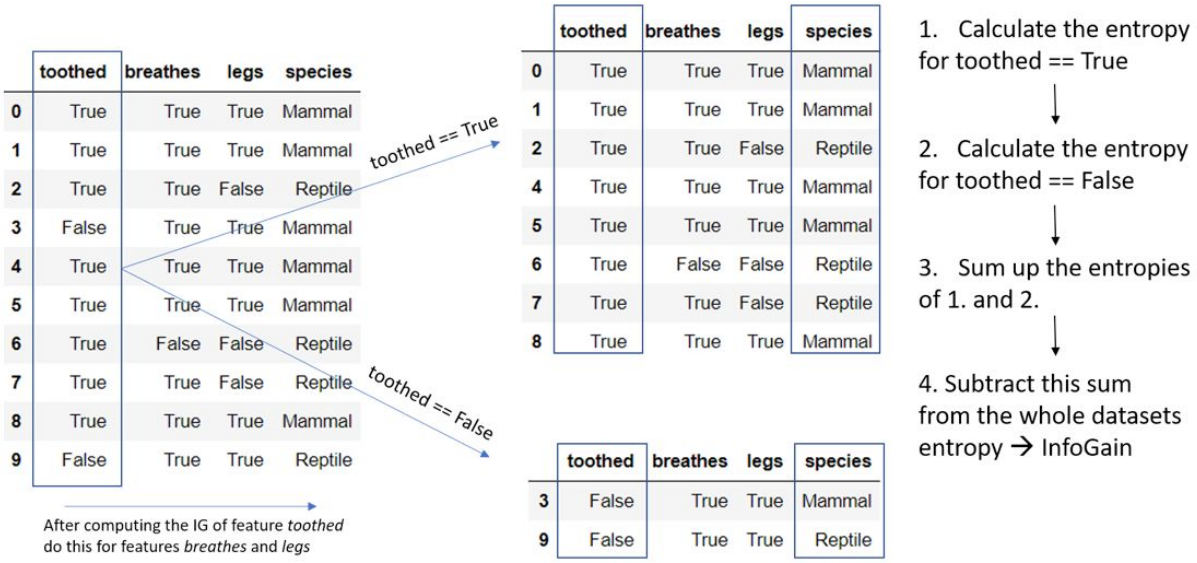
  H(x): $H(x) = -((-0.5) + (-0.464) + (-0.521)) = 1.485$
- For the mammal/reptile example
- $H(x) = -((0.6 * log_2(0.6)) + (0.4 * log_2(0.4))) = 0.971$

- Now we want to find the best feature in terms of information gain[finding the feature which splits the data most accurately along the target feature values]
- How can we check which of the descriptive features most accurately splits the dataset, or remains the dataset with the lowest impurity
- The information gain of a feature is calculated with:
  - $$InfoGain(feature_d) = Entropy(D) - Entropy(feature_d)$$
  - So we have to split the dataset along the values of each feature and then treat these subsets as if they
- Formula for the information gain calculation per feature is:

$$InforGain(feature_d, D) = Entropy(D) - \sum_{t\ \in feature} (\frac{|feature_d = t|}{|D|} * H(feature_d = t))$$

$$= Entropy(D) - \sum_{t\ \in feature} (\frac{|feature_d = t|}{|D|} * (-\sum_{k\ \in target} (P(target = k, feature_d = t) * log_2(P(target = k, feature_d = t)))))$$

- Essentially, for each descriptive feature, we sum up the resulting entropies for splitting the dataset along the feature values and additionally weigh the feature value entropies by their occurrence probability



	toothed	breathes	legs	species
0	True	True	True	Mammal
1	True	True	True	Mammal
2	True	True	False	Reptile
3	False	True	True	Mammal
4	True	True	True	Mammal
5	True	True	True	Mammal
6	True	False	False	Reptile
7	True	True	False	Reptile
8	True	True	True	Mammal
9	False	True	True	Reptile

	toothed	breathes	legs	species
0	True	True	True	Mammal
1	True	True	True	Mammal
2	True	True	False	Reptile
4	True	True	True	Mammal
5	True	True	True	Mammal
6	True	False	False	Reptile
7	True	True	False	Reptile
8	True	True	True	Mammal

	toothed	breathes	legs	species
3	False	True	True	Mammal
9	False	True	True	Reptile

-

$$H(toothed) = (\frac{8}{10}(-((\underbrace{\frac{5}{8} * log2(\frac{5}{8})}_{\text{toothed = True ; Mammal}}) + (\underbrace{\frac{3}{8} * log2(\frac{3}{8})}_{\text{toothed = True ; Reptile}}))) + \frac{2}{10}(-((\underbrace{\frac{1}{2} * log2(\frac{1}{2})}_{\text{toothed = False ; Mammal}}) + (\underbrace{\frac{1}{2} * log2(\frac{1}{2})}_{\text{toothed = False ; Reptile}})))$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{\text{toothed = True}} \qquad\qquad \underbrace{\qquad\qquad\qquad\qquad}_{\text{toothed = False}}$$

= 0.963547

$InfoGain(toothed) = 0.971 - 0.963547 =$ **0.00745**

breathes:

$H(breathes) = (\frac{9}{10} * -((\frac{6}{9} * log_2(\frac{6}{9})) + (\frac{3}{9} * log_2(\frac{3}{9}))) + \frac{1}{10} * -((0) + (1 * log_2(1)))) =$ **0.82647**

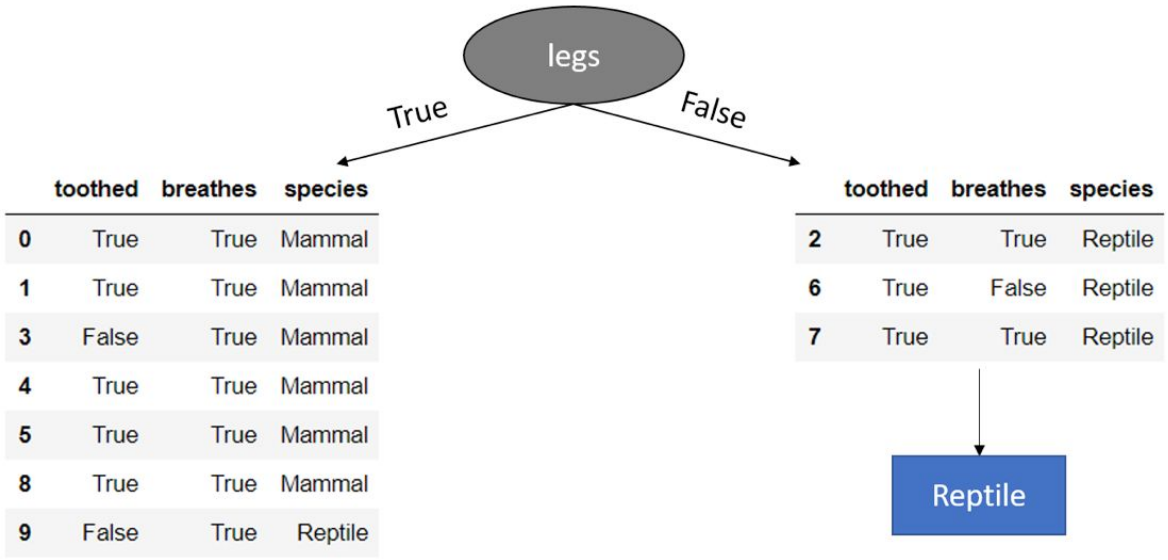
$InfoGain(breathes) = 0.971 - 0.82647 =$ **0.1445**

legs:

$H(legs) = \frac{7}{10} * -((\frac{6}{7} * log_2(\frac{6}{7})) + (\frac{1}{7} * log_2(\frac{1}{7}))) + \frac{3}{10} * -((0) + (1 * log_2(1))) =$**0.41417**

$InfoGain(legs) = 0.971 - 0.41417 =$ **0.5568**

- Splitting the dataset feature of legs results in the largest information gain and we should use this feature for our root node, so now the decision tree looks like this:



	toothed	breathes	species
0	True	True	Mammal
1	True	True	Mammal
3	False	True	Mammal
4	True	True	Mammal
5	True	True	Mammal
8	True	True	Mammal
9	False	True	Reptile

	toothed	breathes	species
2	True	True	Reptile
6	True	False	Reptile
7	True	True	Reptile

-
- Now we look at and compare the two remaining features

  Information gain calculation for the features *toothed* and *breathes* for the remaining dataset *legs* == *True*:

  Entropy of the (new) sub data set after first split:

  $H(D) = -((\frac{6}{7} * log_2(\frac{6}{7})) + (\frac{1}{7} * log_2(\frac{1}{7}))) =$ **0.5917**

  toothed:

  $H(toothed) = \frac{5}{7} * -((1 * log_2(1)) + (0)) + \frac{2}{7} * -((\frac{1}{2} * log_2(\frac{1}{2})) + (\frac{1}{2} * log_2(\frac{1}{2}))) =$ **0.285**

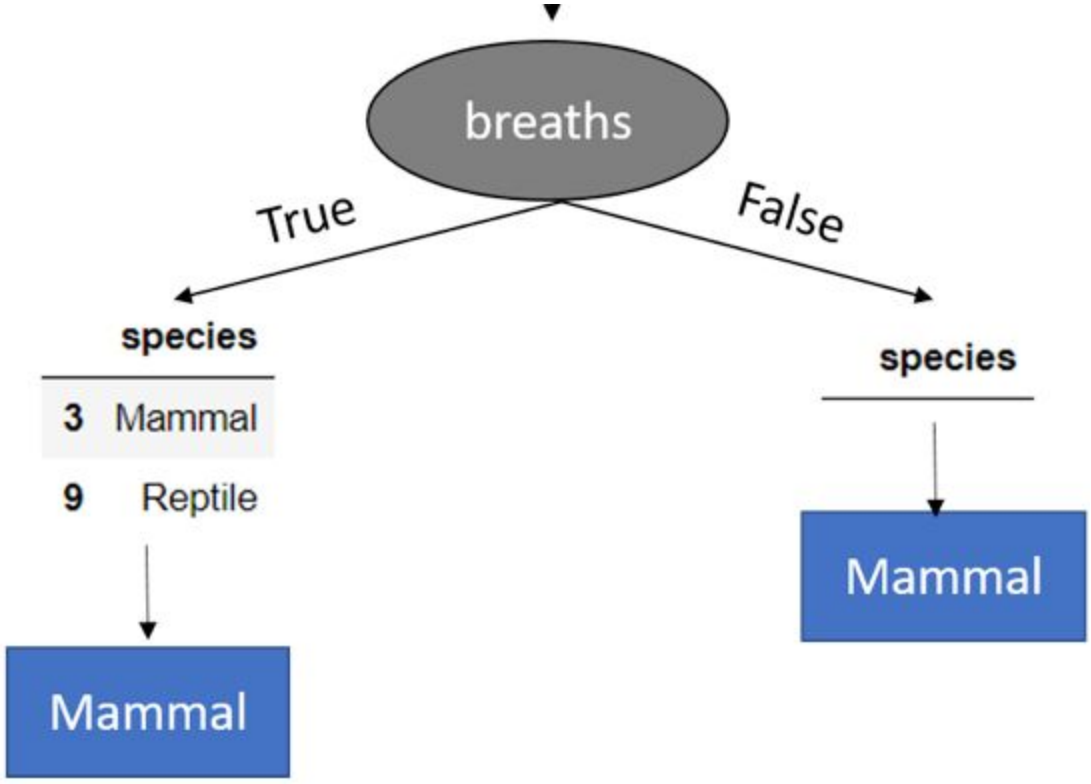
  $InfoGain(toothed) = 0.5917 - 0.285 =$ **0.3067**

  breathes:

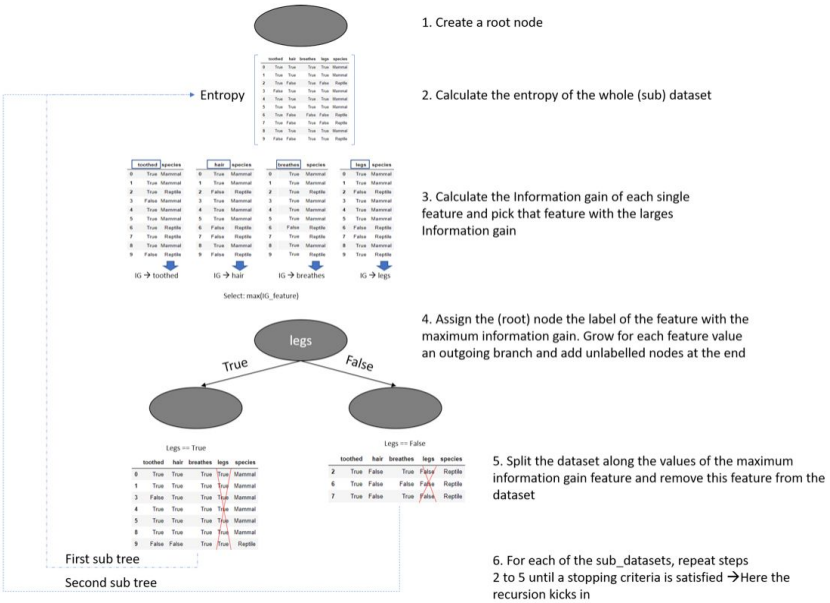
  $H(breathes) = \frac{7}{7} * -((\frac{6}{7} * log_2(\frac{6}{7})) + (\frac{1}{7} * log_2(\frac{1}{7}))) + 0 =$ **5917**

  $InfoGain(toothed) = 0.5917 - 0.5917 =$ **0**

  The dataset for toothed == False still contains a mixture of different target feature values why we proceed partitioning on the last left feature (== breathes)
-
- The decision tree now starts with the comparison of legs, then toothed and then breathes
- The last split looks like this:



- The breathes property solely contains data where breathes == true, so for false there are no instances in the dataset. In that case, we return the most frequently occurring target feature value in the original dataset, which is mammal
- Pseudocode/idea for ID3 algorithm:



- Classification Decision trees from Scratch in Python
- 4 important steps to consider:
  - The calculation of the information gain

    - The recursive call of the TreeModel
    - The building of the actual tree structure
    - The species prediction of a new unseen animal-instance

```
#Import the dataset and define the feature as well as the target datasets / columns#
dataset = pd.read_csv('data/zoo.csv',
                      names=['animal_name','hair','feathers','eggs','milk',
                                           'airbone','aquatic','predator','toothed','backbone',
                                          'breathes','venomous','fins','legs','tail','domestic','catsize','class',])#Import all columns
omitting the fist which consists the names of the animals
```

    - This imports the dataset and defines the features as well as the target datasets

```
def entropy(target_col):
    """
    Calculate the entropy of a dataset.
    The only parameter of this function is the target_col parameter which specifies the target column
    """
    elements,counts = np.unique(target_col,return_counts = True)
    entropy = np.sum([(-counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts)) for i in range(len(elements))])
    return entropy
```

    - We define a function to calculate the entropy of a dataset

```
def InfoGain(data,split_attribute_name,target_name="class"):
    """
    Calculate the information gain of a dataset. This function takes three parameters:
    1. data = The dataset for whose feature the IG should be calculated
    2. split_attribute_name = the name of the feature for which the information gain should be calculated
    3. target_name = the name of the target feature. The default for this example is "class"
    """
    #Calculate the entropy of the total dataset
    total_entropy = entropy(data[target_name])

    ##Calculate the entropy of the dataset

    #Calculate the values and the corresponding counts for the split attribute
    vals,counts= np.unique(data[split_attribute_name],return_counts=True)

    #Calculate the weighted entropy
    Weighted_Entropy = np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_name]==vals[i]).dropna()[target_name]) for i i
n range(len(vals))])

    #Calculate the information gain
    Information_Gain = total_entropy - Weighted_Entropy
    return Information_Gain
```

    - We define a function to calculate the information gain of a dataset
- **This is writing from scratch, but there are libraries to use**
- ***Using sklearn***
    - The steps to use the sklearn classification follow the principal sklearn API which are:
        - Choose the model you want to use → DecisionTreeClassifier/Regressor
        - Set the model hyperparameters → i.e number of minimum samples per leaf
        - Create a feature data set as well as a target array containing the labels for the instances
        - Fit the model to the training data
        - Use the fitted model on unseen data

- ```
  #Import the DecisionTreeClassifier
  from sklearn.tree import DecisionTreeClassifier

  ################################################################################

  ################################################################################

  """
  Import the Zoo Dataset
  """

  #Import the dataset
  dataset = pd.read_csv('data/zoo.csv')
  #We drop the animal names since this is not a good feature to split the data on
  dataset=dataset.drop('animal_name',axis=1)
  ```
- ```
  """
  Split the data into a training and a testing set
  """

  train_features = dataset.iloc[:80,:-1]
  test_features = dataset.iloc[80:,:-1]
  train_targets = dataset.iloc[:80,-1]
  test_targets = dataset.iloc[80:,-1]
  ```
- ```
  """
  Train the model
  """

  tree = DecisionTreeClassifier(criterion = 'entropy').fit(train_features,train_targets)
  ```
- ```
  """
  Predict the classes of new, unseen data
  """
  prediction = tree.predict(test_features)
  ```
- ```
  """
  Check the accuracy
  """

  print("The prediction accuracy is: ",tree.score(test_features,test_targets)*100,"%")
  ```

● Implementation Example w/ Regression Trees

- Using this Boston Housing Dataset [Boston Housing]
  - Contains 506 samples and 14 attributes
  - For simplicity, only using MEDV[median value of owner-occupied purposes] as the target and LSTAT[percentage of lower status of the population]
  ```

    - Target - the variable whose values are to be modeled and predicted by other variables
    - Features - variables that are usually characteristics or properties
- First, import the necessary attributes from *scikit-learn* into a pandas DataFrame

```
import pandas as pd
from sklearn import datasets

boston = datasets.load_boston()             # Load Boston Dataset
df = pd.DataFrame(boston.data[:, 12])       # Create DataFrame using only the LSAT feature
df.columns = ['LSTAT']
df['MEDV'] = boston.target                  # Create new column with the target MEDV
df.head()
```

	LSTAT	MEDV
0	4.98	24.0
1	9.14	21.6
2	4.03	34.7
3	2.94	33.4
4	5.33	36.2

- Next, we use the DecisionTreeRegressor in *scikit-learn* to train a regression tree

```
from sklearn.tree import DecisionTreeRegressor    # Import decision tree regression mode

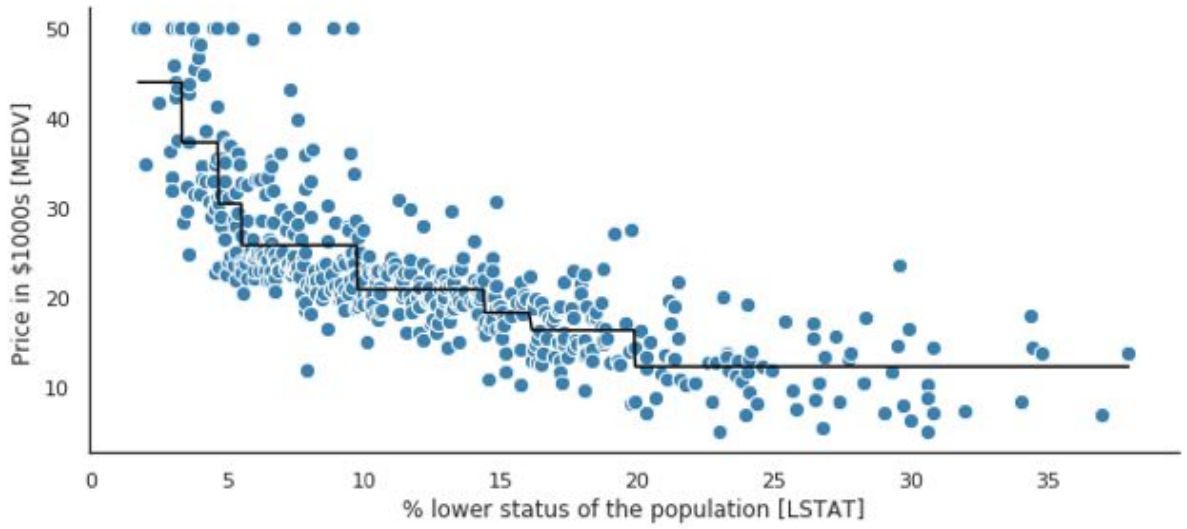
X = df[['LSTAT']].values                          # Assign matrix X
y = df['MEDV'].values                             # Assign vector y

sort_idx = X.flatten().argsort()                  # Sort X and y by ascending values of
X = X[sort_idx]
y = y[sort_idx]

tree = DecisionTreeRegressor(criterion='mse',     # Initialize and fit regressor
                             max_depth=3)
tree.fit(X, y)
```

    - Entropy as a measure of impurity is a useful criteria for classification, but for regression trees, we need an impurity metric that is suitable for continuous variables, so we use **weighted mean squared error[MSE]** instead
- We then model the relationship between MEDV and LSTAT to see what the line fit of a regression tree looks like

```
plt.figure(figsize=(16, 8))
plt.scatter(X, y, c='steelblue',                    # Plot actual target against features
            edgecolor='white', s=70)
plt.plot(X, tree.predict(X),                        # Plot predicted target against featur
         color='black', lw=2)
plt.xlabel('% lower status of the population [LSTAT]')
plt.ylabel('Price in $1000s [MEDV]')
plt.show()
```



Another Decision Tree Regression Example w/ sklearn

- *Decision tree regression observes the features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output
  - Continuous output means that the output/result is not discrete
  - Discrete Output Example: a weather prediction model that predicts whether or not there will be rain on a given day
  - Continuous Output Example: a profit prediction model that **states the probable profit that can be generated from the sale of a product**
- Step 1: import the required libraries

```
# import numpy package for arrays and stuff
import numpy as np

# import matplotlib.pyplot for plotting our result
import matplotlib.pyplot as plt

# import pandas for importing csv files
import pandas as pd
```

- Step 2: Initialize and print[optional] dataset. For this step, you can either MANUALLY initialize the array OR open up a csv file to read the data [label, production cost, profit]

```
dataset = np.array(
[['Asset Flip', 100, 1000],
['Text Based', 500, 3000],
['Visual Novel', 1500, 5000],
['2D Pixel Art', 3500, 8000],
['2D Vector Art', 5000, 6500],
['Strategy', 6000, 7000],
['First Person Shooter', 8000, 15000],
['Simulator', 9500, 20000],
['Racing', 12000, 21000],
['RPG', 14000, 25000],
['Sandbox', 15500, 27000],
['Open-World', 16500, 30000],
['MMOFPS', 25000, 52000],
['MMORPG', 30000, 80000]
])

# print the dataset
print(dataset)
```

OR

```
# import dataset
# dataset = pd.read_csv('Data.csv')
# alternatively open up .csv file to read data
```

- Step 3: select all the rows and column 1 from the dataset and assign to “X”

```
# select all rows by : and column 1
# by 1:2 representing features
X = dataset[:, 1:2].astype(int)

# print X
print(X)
```

```
[[  100]
 [  500]
 [ 1500]
 [ 3500]
 [ 5000]
 [ 6000]
 [ 8000]
 [ 9500]
 [12000]
 [14000]
 [15500]
 [16500]
 [25000]
 [30000]]
```

-
- Step 4: Select all of the rows and column 2 from dataset and assign to “Y”

```
# select all rows by : and column 2
# by 2 to Y representing labels
y = dataset[:, 2].astype(int)

# print y
print(y)
```

-
- Step 5: Fit Decision Tree Regressor to the Dataset

```
# import the regressor
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
regressor = DecisionTreeRegressor(random_state = 0)

# fit the regressor with X and Y data
regressor.fit(X, y)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=0, splitter='best')
```

- Step 6: Predicting a new value [this seems to be inputting 3750 as production cost, and we are predicting the y axis, which is the predicted profit. So this code predicts that if we spend 3750 on production costs, about 8000 will be the total profit amount]

Step 6: Predicting a new value

```
# predicting a new value

# test the output by changing values, like 3750
y_pred = regressor.predict(3750)

# print the predicted price
print("Predicted price: % d\n"% y_pred)
```

- Step 7: visualizing the result

```
# arange for creating a range of values
# from min value of X to max value of X
# with a difference of 0.01 between two
# consecutive values
X_grid = np.arange(min(X), max(X), 0.01)

# reshape for reshaping the data into
# a len(X_grid)*1 array, i.e. to make
# a column out of the X_grid values
X_grid = X_grid.reshape((len(X_grid), 1))

# scatter plot for original data
plt.scatter(X, y, color = 'red')

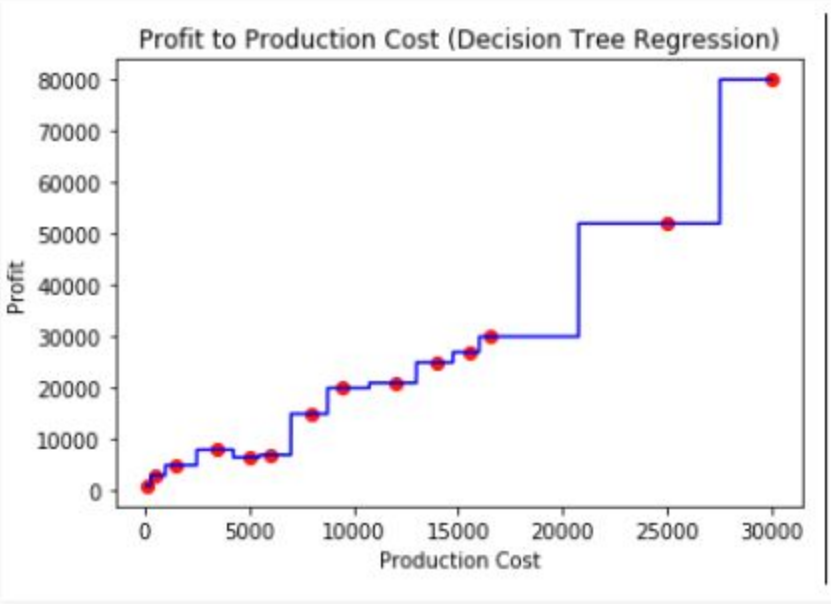
# plot predicted data
plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')

# specify title
plt.title('Profit to Production Cost (Decision Tree Regression)')

# specify X axis label
plt.xlabel('Production Cost')

# specify Y axis label
plt.ylabel('Profit')

# show the plot
plt.show()
```

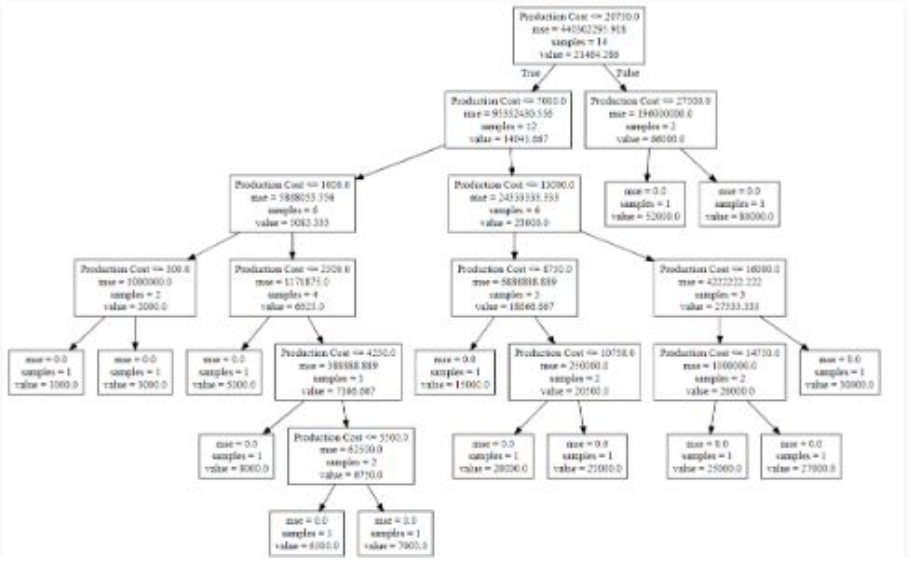


- Step 8: exporting tree and showing the decision tree [this uses http://www.webgraphviz.com/ by copying the data from the “tree.dot” file]

```
# import export_graphviz
from sklearn.tree import export_graphviz

# export the decision tree to a tree.dot file
# for visualizing the plot easily anywhere
export_graphviz(regressor, out_file ='tree.dot',
                feature_names =['Production Cost'])
```

Output (Decision Tree):



● Ideas/Notes from Adisak

- We are most likely looking at a multi target regression → multiple dependent variables based on inputs
- Look at columns that could be the target of something we want to predict
- We need good features to predict targets
- More so focused on cleaning up the data, a smaller portion of the project looks at machine learning
- Around 300 doors, each door has about 300 parameters a piece
- Steps:
  - First step: talk to client, learn the process of making the batch
  - Figure out how to rearrange the batches to align with respective outputs
  - Understand batch file, clean up batch file, select relevant features
    - Record, one batch?

- Is the manufacturing process strict, or are there steps we can skip
- Consider naming the steps in the manufacturing process
- Too many features, throwing everything in, may skew the learning model
    - Ask about each of the parameters and see if they are important
    - EX: process 1 [say 5 features] columns, each row is a door?
- Target predictions [moisture should be minimized]
    - Regression on Flexural, moisture, density, compression (we will need 4 of these)
    - Need humidities to correspond to each of the test dates
- https://machinelearningmastery.com/calculate-feature-importance-with-python/
- Calculate feature importance, find those values
    - That's essentially the answer, those features have the most impact
- Then we can visualize that and make a list ???
- https://mljar.com/blog/feature-importance-in-random-forest/
- CoLab?
- Data Studio?