wbs
WARWICK BUSINESS SCHOOL
THE UNIVERSITY OF WARWICK

For the Change Makers

Advanced Programming for Data Science

Week 8: Data Analysis and Modeling
Information Systems and Management
Warwick Business School

Supervised learning: regression vs. classification

- A regression model predicts continuous values. For example, regression models make predictions that answer questions like the following:
- What is the value of a house in California?
- What is the probability that a user will click on this ad?
- A classification model predicts discrete values. For example, classification models make predictions that answer questions like the following:
- Is a given email message spam or not spam?
- Is this an image of a dog, a cat, or a hamster?

What is classification?

- Determining to what group a data element belongs
- Based on data contained within that element
  - Or “attributes” of that “entity”
- Examples
  - Determining whether a customer should be given a loan
  - Flagging a credit card transaction as a fraudulent charge
  - Categorizing a news story as finance, entertainment, or sports

Goals

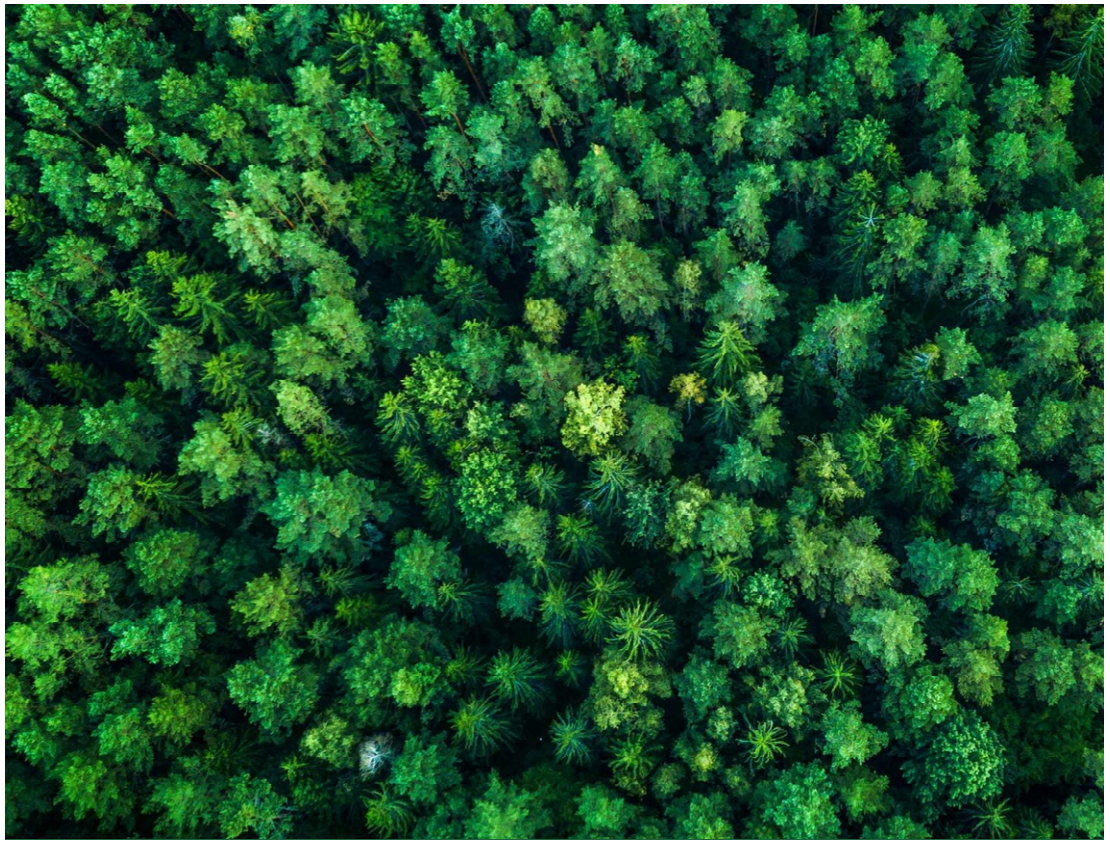
- The model developed using the **training** set should assign **test** set records to the right category
- It won’t be 100% accurate, but should be as close as possible
- Once this is done, the model’s rules can be applied to new records as they come along
- We want an automated, reliable way to predict the outcome

How classification works

1. Choose a set of records for the training set
2. Choose a set of records for the test set
3. Choose a class attribute (classifier)
4. Find model that predicts the class attribute as a function of the other attributes
5. Apply that model to the test set to check accuracy
6. Apply the final model to future records to classify

Tree-based machine learning algorithms

- Decision tree classification
- Random Forest
- Boosting Tree
- Tree regression



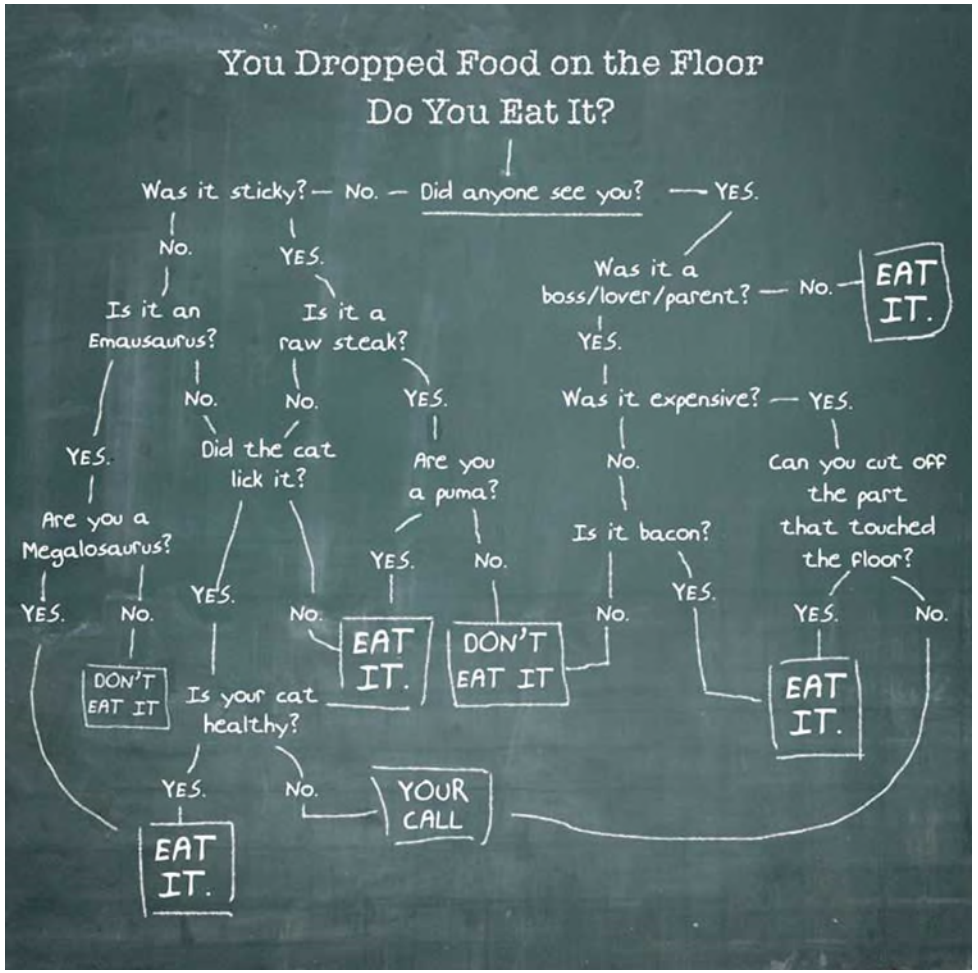
Some classification algorithms

	Speed	Accuracy	Explainability	Data Size
Naïve Bayes	Faster	Lower	Lower	Smaller
Logistic Regression	Faster	Lower	Higher	Smaller
Linear SVM	Faster	Lower	Lower	Larger
Kernel SVM	Slower	Higher	Lower	Larger
Decision Tree	Faster	Lower	Higher	Larger
Random Forest	Slower	Higher	Lower	Larger
Gradient Boosting Tree	Slower	Higher	Lower	Larger
Neural Network	Slower	Higher	Lower	Larger

Decision Tree

Decision Trees

- Used to classify data according to a pre-defined outcome
- Based on characteristics of that data
- Uses
  - Predict whether a customer should receive a loan
  - Flag a credit card charge as legitimate
  - Determine whether an investment will pay off



http://www.mindtoss.com/2010/01/25/five-second-rule-decision-chart/

Decision Tree Learning

Training Set

Trans. ID	Charge Amount	Avg. Charge 6 months	Item	Same state as billing	Classification
1	$800	$100	Electronics	No	Fraudulent
2	$60	$100	Gas	Yes	Legitimate
3	$1	$50	Gas	No	Fraudulent
4	$200	$100	Restaurant	Yes	Legitimate
5	$50	$40	Gas	No	Legitimate
6	$80	$80	Groceries	Yes	Legitimate
7	$140	$100	Retail	No	Legitimate
8	$140	$100	Retail	No	Fraudulent

Test Set

Trans. ID	Charge Amount	Avg. Charge 6 months	Item	Same state as billing	Classification
101	$100	$200	Electronics	Yes	?
102	$200	$100	Groceries	No	?
103	$1	$100	Gas	Yes	?
104	$30	$25	Restaurant	Yes	?

Derive model

Classification software

Apply model

Example: Credit Card Default

Predictors | Classification

TID	Income	Debt	Owns/ Rents	Outcome
1	25k	35%	Owns	**Default**
2	35k	40%	Rent	**Default**
3	33k	15%	Owns	**No default**
4	28k	19%	Rents	**Default**
5	55k	30%	Owns	**No default**
6	48k	35%	Rent	**Default**
7	65k	17%	Owns	**No default**
8	85k	10%	Rents	**No default**
…				

Training Data

We create the tree from a set of training data

Each unique combination of predictors is associated with an outcome

This set was “rigged” so that every combination is accounted for and has an outcome

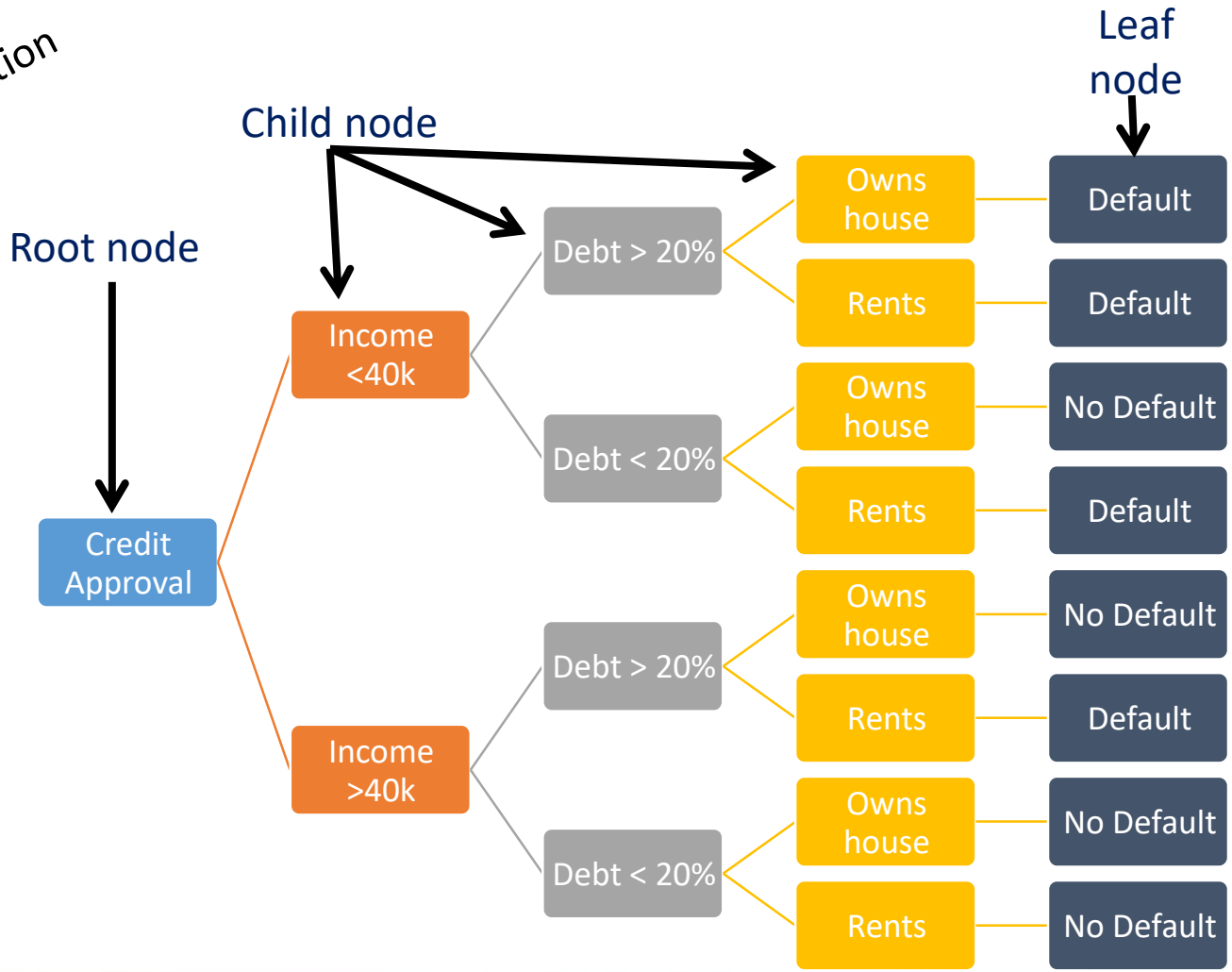
Example: Credit Card Default

Predictors

Classification

TID	Income	Debt	Owns/ Rents	Outcome
1	25k	35%	Owns	**Default**
2	35k	40%	Rent	**Default**
3	33k	15%	Owns	**No default**
4	28k	19%	Rents	**Default**
5	55k	30%	Owns	**No default**
6	48k	35%	Rent	**Default**
7	65k	17%	Owns	**No default**
8	85k	10%	Rents	**No default**
…				

Training Data



Same Data, Different Tree

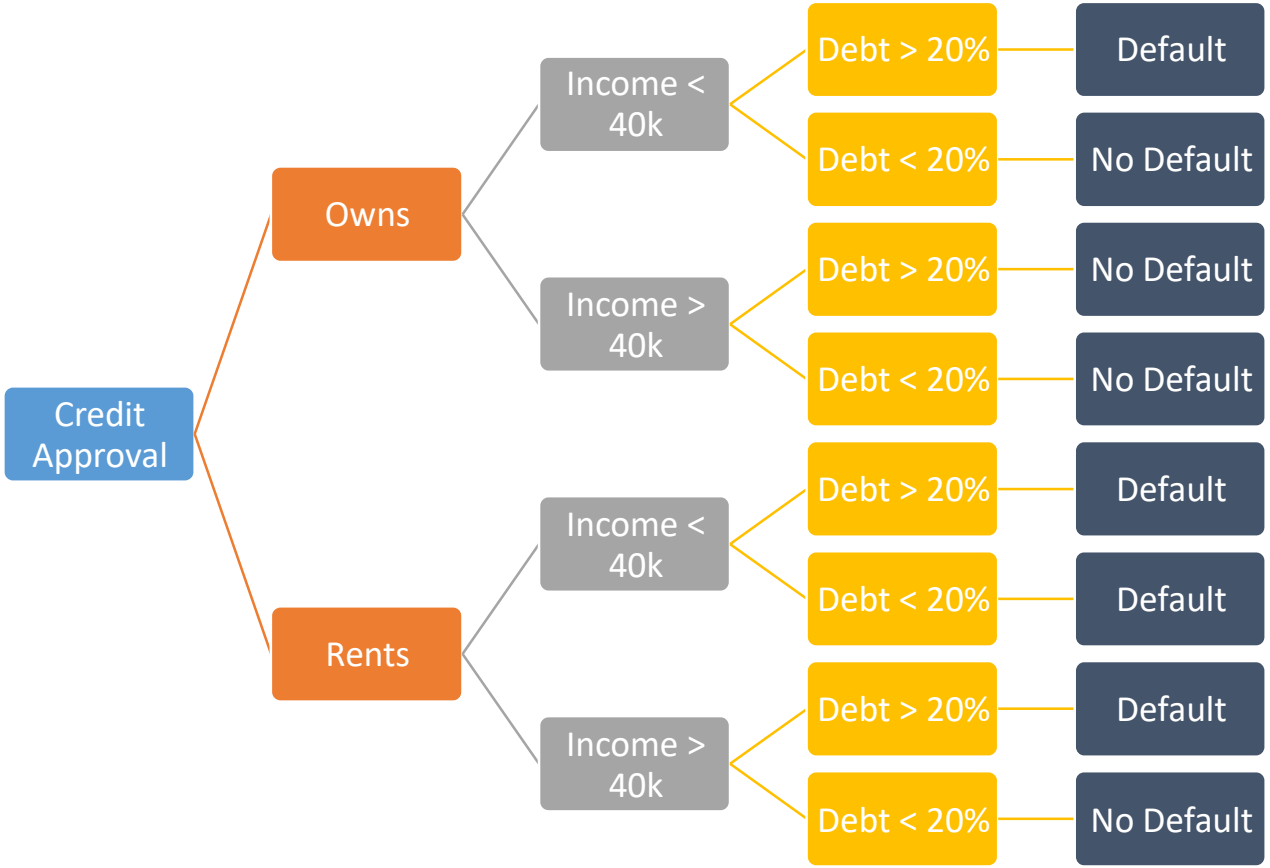
Predictors

Classification

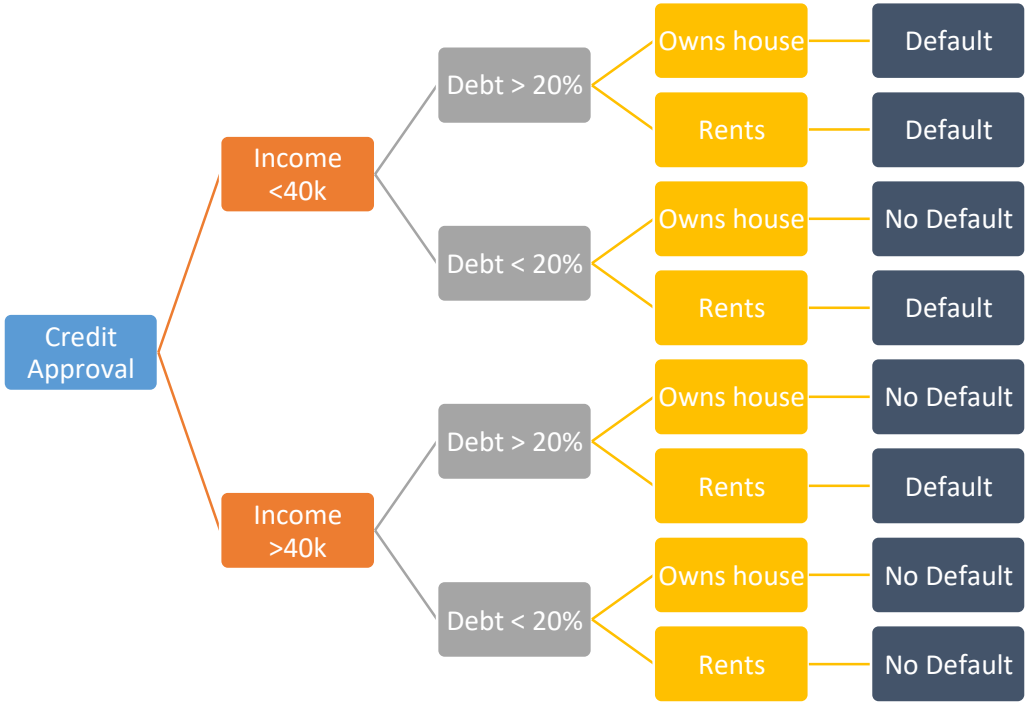
TID	Income	Debt	Owns/ Rents	Outcome
1	25k	35%	Owns	**Default**
2	35k	40%	Rent	**Default**
3	33k	15%	Owns	**No default**
4	28k	19%	Rents	**Default**
5	55k	30%	Owns	**No default**
6	48k	35%	Rent	**Default**
7	65k	17%	Owns	**No default**
8	85k	10%	Rents	**No default**
…				

Training Data

We just changed the order of the predictors.



Apply the data to new (test) data



TID	Income	Debt	Owns/ Rents	Decision (Predicted)	Decision (Actual)
1	80k	35%	Rent	**Default**	**No Default**
2	20k	40%	Owns	**Default**	**Default**
3	15k	15%	Owns	**No Default**	**No Default**
4	50k	19%	Rents	**No Default**	**Default**
5	35k	30%	Owns	**Default**	**No Default**

Test Data

How well did the decision tree do in predicting the outcome?

Issues

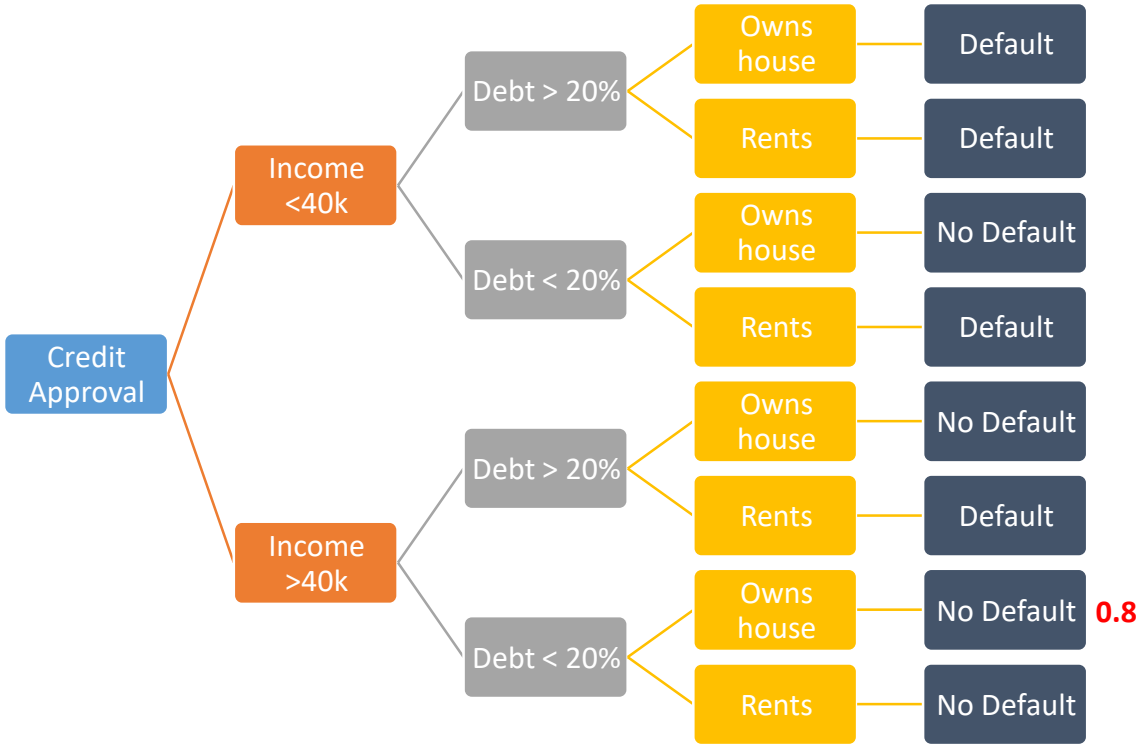
- In reality, the software has to deal with several issues
  - When the same set of predictors result in different outcomes
  - When not every combination of predictors is in the training set
  - When multiple paths result in the same outcome

Tree Induction Algorithms

Tree induction algorithms take large sets of data and compute the tree

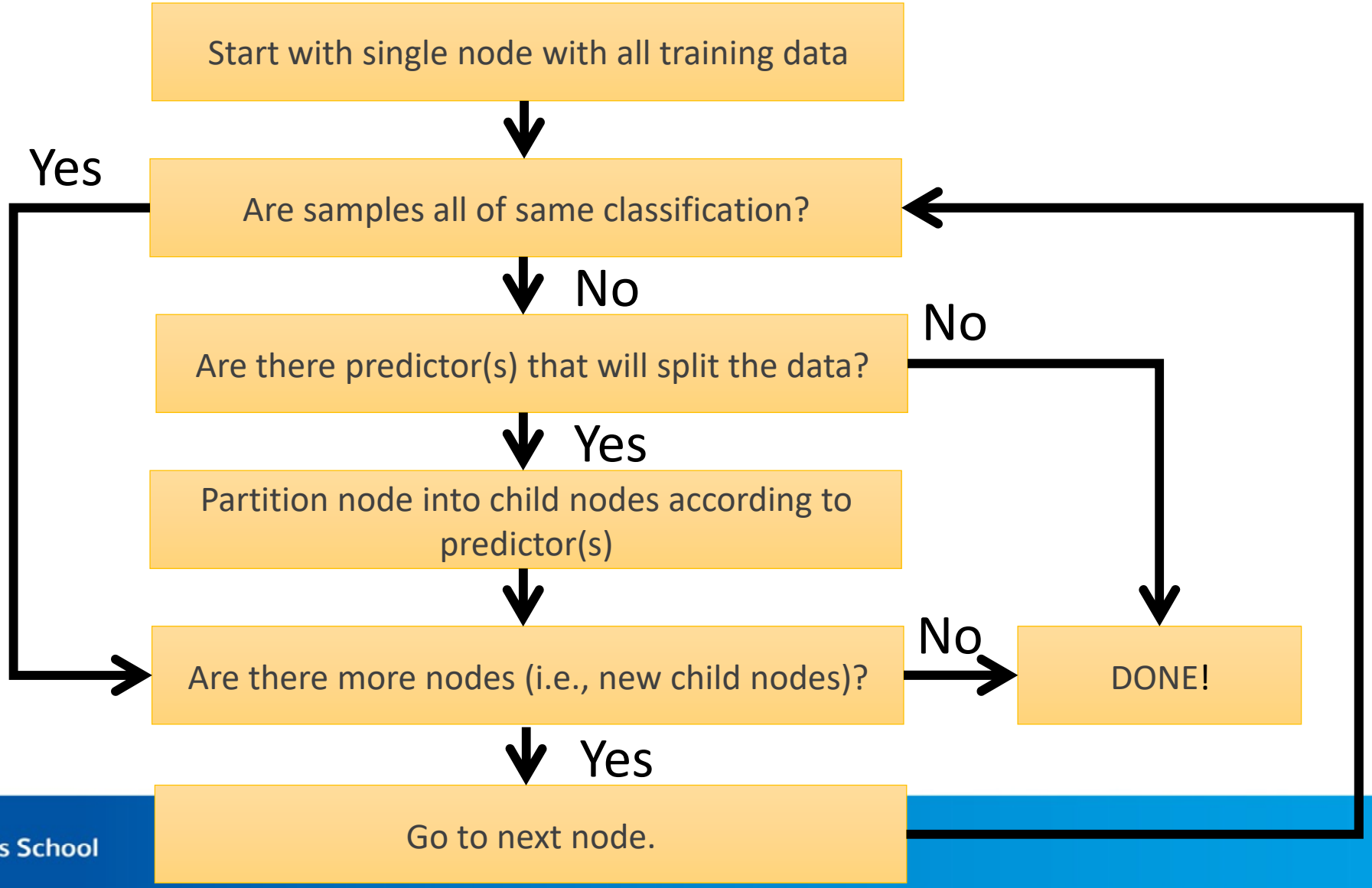
Similar cases may have different outcomes

So probability of an outcome is computed

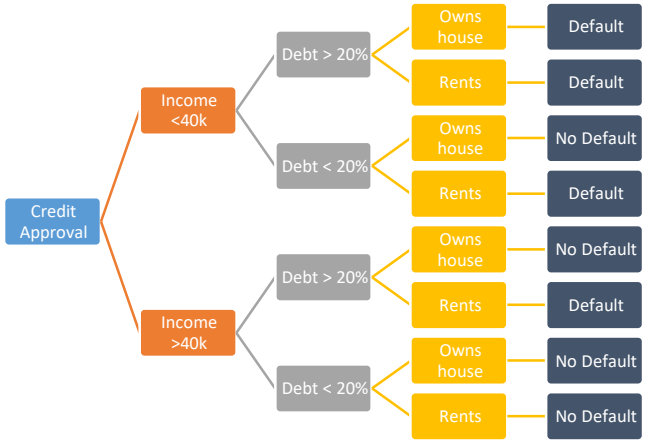


For instance, you may find:
When income > 40k, debt < 20%, and the customers rents no default occurs 80% of the time.

How the induction algorithm works



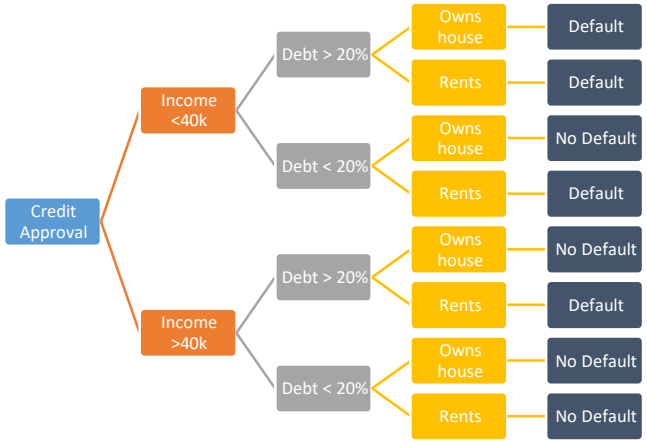
Start with root node

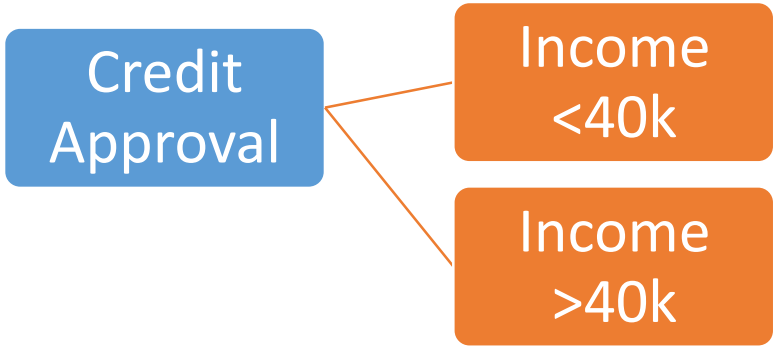


Credit Approval

- There are both “defaults” and “no defaults” in the set
- So we need to look for predictors to split the data

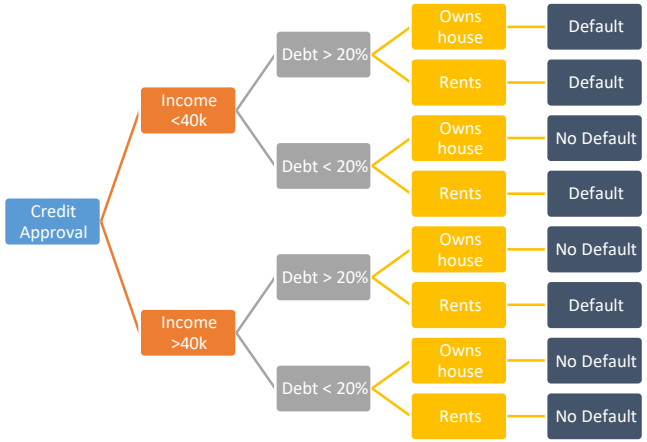
Split on income

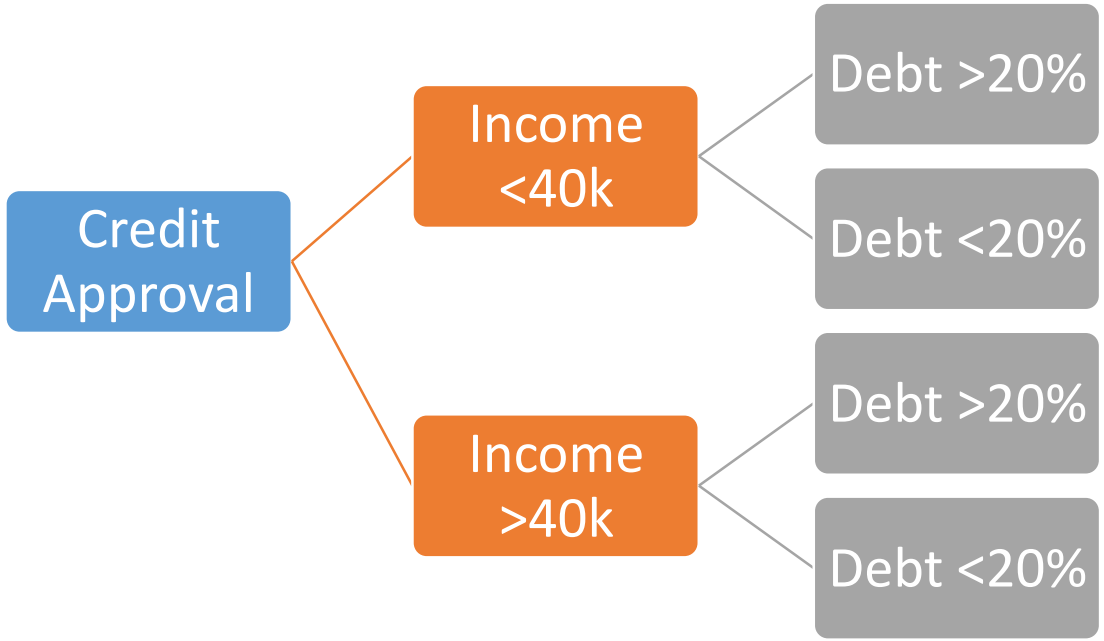




- Income is a factor
  - (Income < 40, Debt > 20, Owns = Default)
    but
    (Income > 40, Debt > 20, Owns = No default)
- But there are also a combination of defaults and no defaults within each income group
- So look for another split

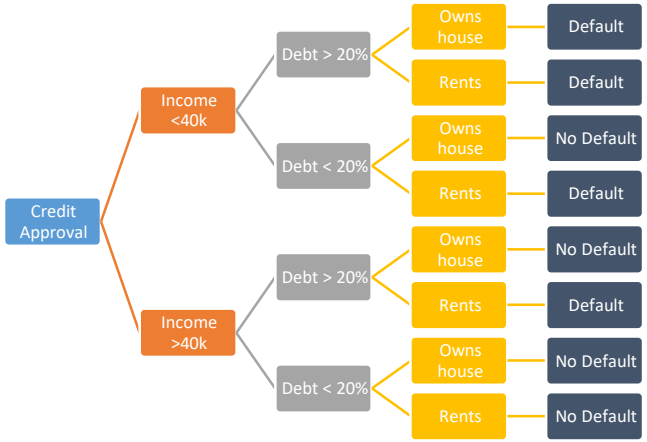
Split on debt

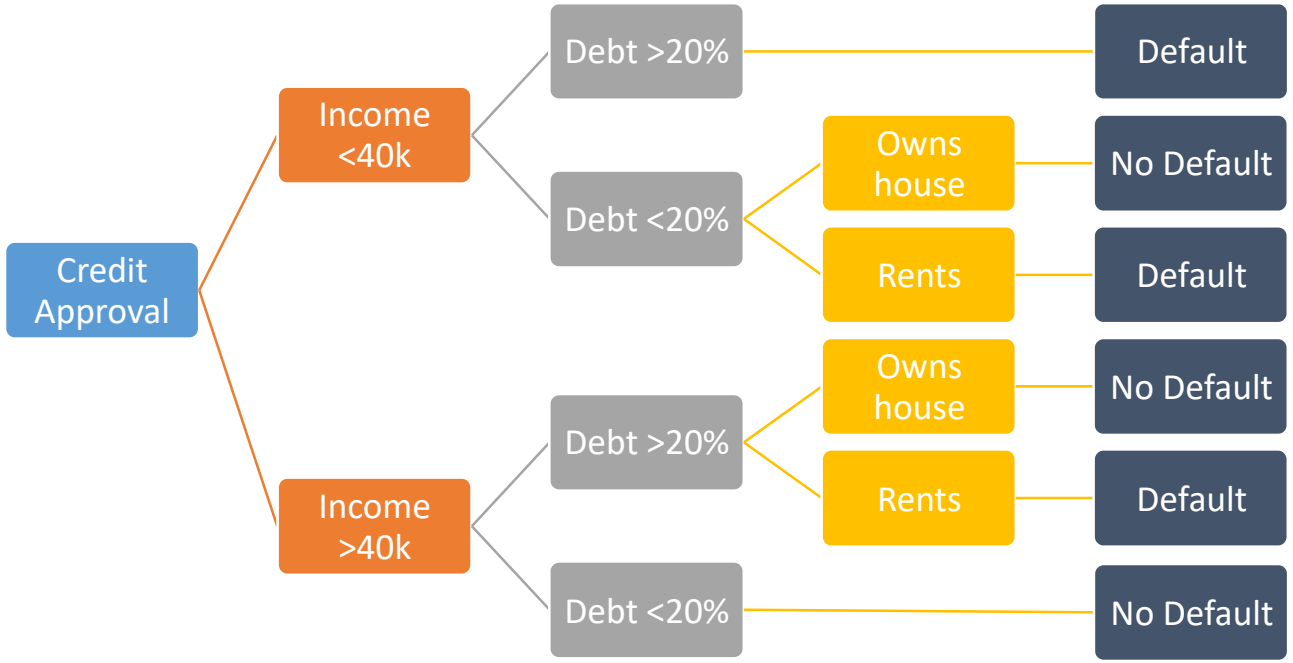




- Debt is a factor
  - (Income < 40, Debt < 20, Owns = No default)
    but
    (Income < 40, Debt < 20, Rents = Default)
- But there are also a combination of defaults and no defaults within some debt groups
- So look for another split

Split on Owns/Rents





- Owns/Rents is a factor
- For some cases it doesn't matter, but for some it does
- So you group similar branches
- …And we stop because we're out of predictors!

How does it know when and how to split?

- Our example was simple because we had one case per branch
- So there was 100% agreement within every branch
- When you have a lot of data, you'll get different outcomes with the same predictors
- So you need criteria to decide when to split the branch using a predictor
- And you need a way of figuring out when your leaf nodes
  - Describe primarily one type of outcome
  - Are different from one another

Splitting method:

- Gini
- Entropy

Advantages and Disadvantages of Trees

- ✓ Easy to explain.
- ✓ Intuitive
- ✓ Graphical representation
- ✓ Easy to deal with categorical variables
- ✓ Less influenced by missing data and outliers
- ❖ Lower accuracy
- ❖ Overfitting

Machine Learning Pipeline

1. create training/testing sets,
2. instantiate the classifier,
3. train the classifier,
4. make predictions,
5. evaluate performance,
6. tweak parameters.

Create training/testing sets

- Make sure your data has been cleaned and in the right format.
- Split your dataset into features and labels.

```
feature_cols = ['Pclass', 'Male', 'Fare']
X = df_titan[feature_cols] # Features
y = df_titan.Survived # Labels
```

Create training/testing sets

- Split your dataset into training and testing sets. This can be done through train_test_split function from model_selection module of sklearn.
  - At least one dataset to be split
  - *test_size*: 0 to 1, the portion to be assigned as the testing dataset.
  - *random_state*: seed used by the random number generator

```
from sklearn.model_selection import train_test_split
# 70% training and 30% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

Instantiating the Classifier

```
# Import Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
# Create Decision Tree classifier object
clf = DecisionTreeClassifier()
```

Train the Classifier

```
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
```

Make Predictions

```
#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Model Computation in sklearn

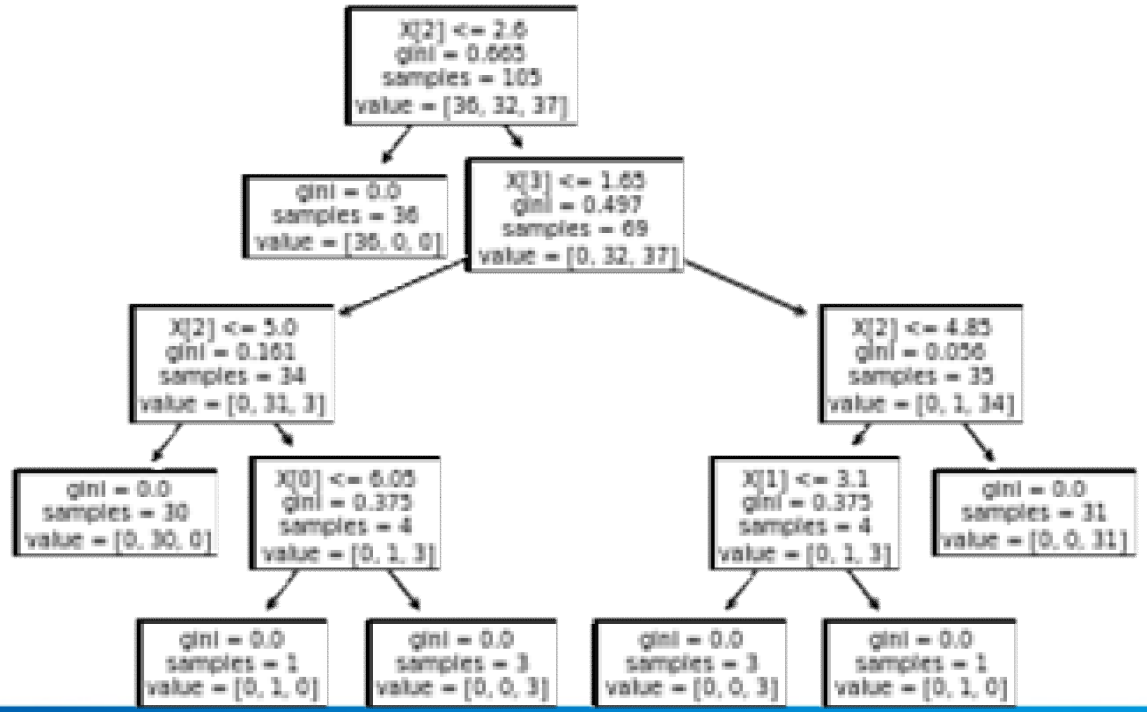
- Generally, the task of machine learning is done in two steps with SK-Learn:

1. Learning (computing): it is usually done with the `fit()` method, which is to compute the model's parameters based on your training data.
2. Predicting: it is usually done with the `predict()` method, which is to make the prediction based on the computed model from previous step.
2. *Transformation: it is usually done with the `transform()` method, which is to convert your data to certain form based on the parameters generated from*

- In most cases, you can combine the two steps using `fit_predict()` method (or `fit_transform()` like our earlier examples).

Evaluate Performance

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Visualize the Tree

```
from sklearn import tree
tree.plot_tree(clf) # added in version 0.21.3
```