INFO5002: Intro to Python for Info Sys

Decision Trees

DSfS 215-225

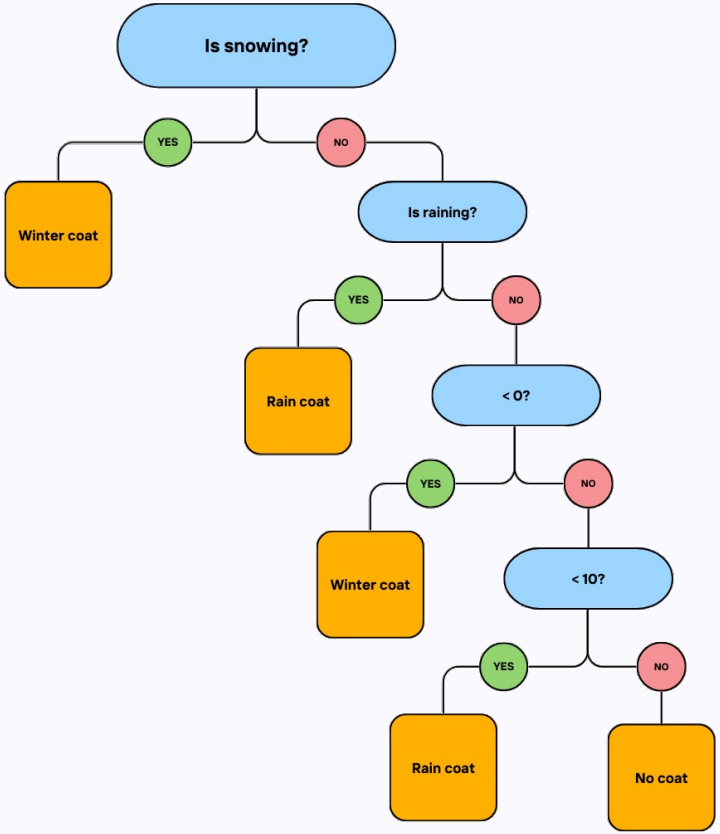
Slides created by: Zachary Doucet

LVX VERITAS VIRTVS
Northeastern University

How do we make decisions?

- Usually we start by looking at one variable and then based on some condition you either do something or another thing.
- If it is snowing, I would wear a winter jacket; if it is raining, a rain jacket; if it is neither but below 0C, I would wear a winter jacket; if below 10C, a rain jacket; if not anything, then no jacket.

As a decision tree



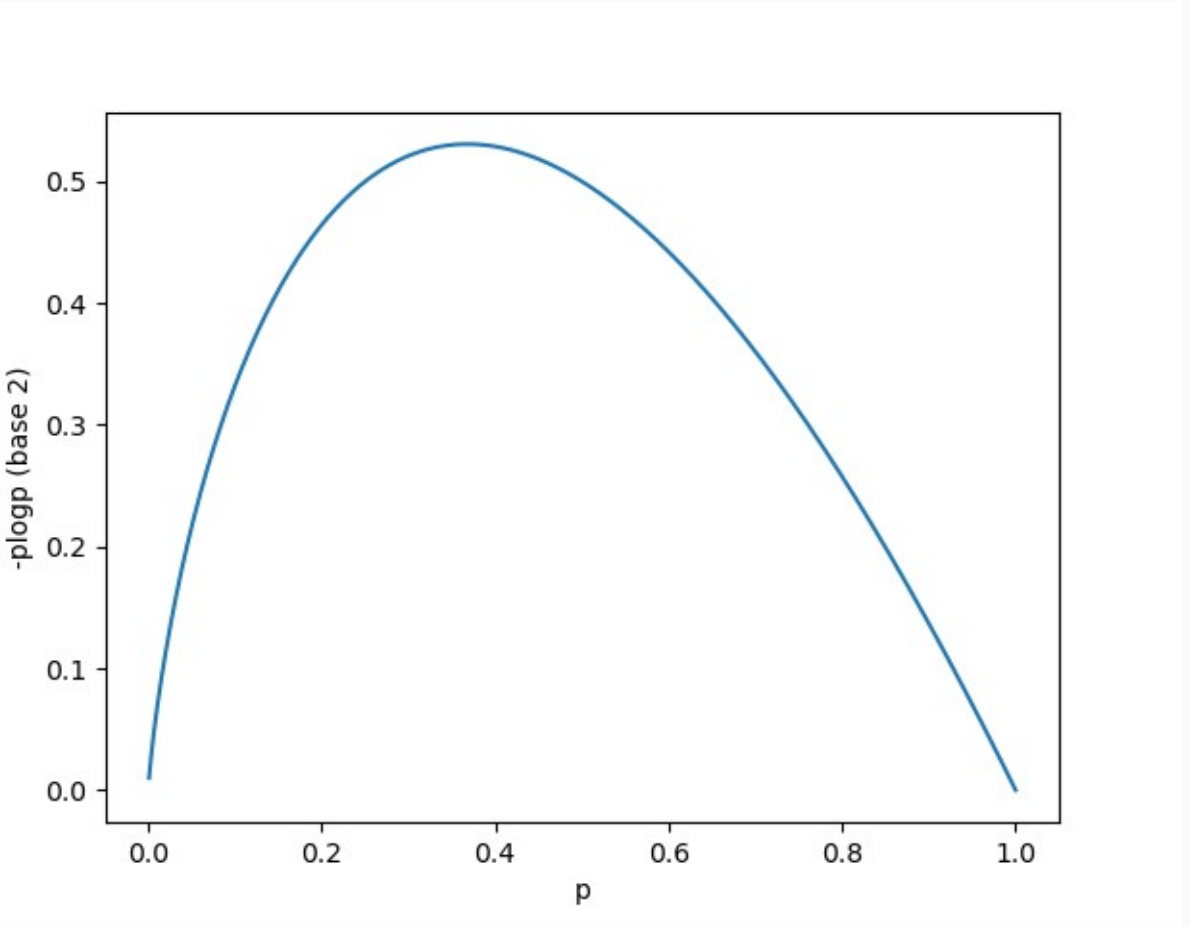
But not manually!

- What if we create a learned algorithm to create these decision trees and create decisions by itself?!
- Nice because then we can give the decisions to a human (human readable), to evaluate.
- Works on numeric and categorical features. No need to numerically encode.

We want good decisions.

- Good decisions are able to split up possibilities in a balanced way. Think of Akinator.
- We can use entropy to represent how spread the data is. p_i is proportion of class_i. The more spread, the less certain.

$$H(s) = -p_1 \log_2 p_1 - \ldots - p_n \log_2 p_n$$




```
def entropy(ps):
    return np.sum(-ps * np.log2(ps))
```

To know how good a split is

- We can weighted sum the entropies of each subset.

  Where q_i is the proportion of samples in S_i.

$$H = q_1 H(S_1) + \ldots + q_m H(S_m)$$

- Smaller is better.

ID3

- If data all have same label => create leaf node of that label.
- If list of attributes (what to split on) is empty => create leaf node that predicts the most popular label.
- If not empty try to partition by each attribute.
  - Choose the one with lowest entropy.
  - Add decision node based on attribute.
  - Recurse on each partitioned subset on remaining attributes.

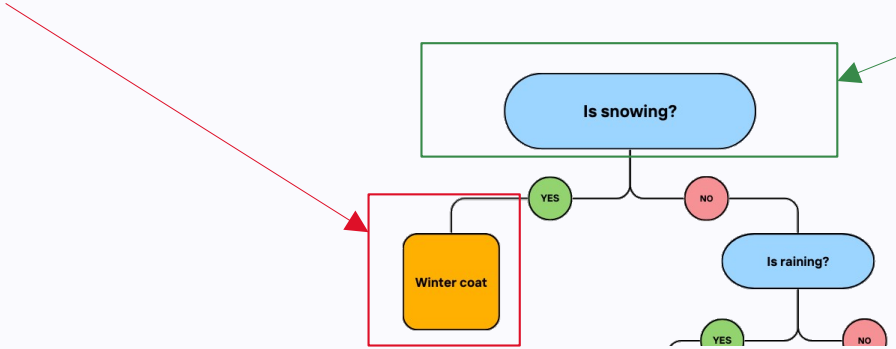
Basic data structures

```
class Leaf:
    def __init__(self, val):
        self.val = val
```

Represents our possible output

```
class Node:
    def __init__(self, key):
        self.key = key
        self.children = {}
```

Represents our question we hold



```
def build_tree(self, data, attributes):
    if len(data) == 0:
       return self.default # no data
    if len(data.unique()) == 1:
       return Leaf(data["label"][0]) # all the same (lowest entropy)
    if len(attributes) == 0:
        return Leaf(data["label"].mode())

    entropies = [self.entropy_partition_by_attribute(data, attribute)
         for attribute in attributes]
    min_idx = entropies.index(min(entropies))
    best_attribute = attributes[min_idx]
    children = self.split_by_attribute(data, best_attribute)
    node = Node(best_attribute) # create a new decision node
    new_attributes = [attribute for attribute
         in attributes if attribute ≠ best_attribute] # remove what we used
   # recurse over each child
    for child in children:
        node.children[child] = self.build_tree(children[child], new_attributes)

    return node
```

Numerical hack

- Given that two numbers have infinite points between them you can subdivide the max – min by a certain amount as your possible classes.

Overview

- Decision trees are nice because they are easily explainable and human readable.
- They can work with both categorical and numerical values.
- They have built in feature selection.
- They struggle with overfitting.

Scikit-Learn

- Need to decide between regression or classification
  - DecisionTreeRegressor: when target continuous value
  - DecisionTreeClassifier: when target is a class

Regressor

```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor(criterion="squared_error",
    splitter="best", max_depth=None, min_samples_split=2,
    min_samples_leaf=1, min_weight_fraction_leaf=0.0,
    max_features=None, random_state=None, max_leaf_nodes=None,
    min_impurity_decrease=0.0, ccp_alpha=0.0,
    monotonic_cst=None)
# To train the model
model.fit(X_train, y_train)

# Get path of decision making (num_samples, num_nodes)
path = model.decision_path(x)

# Get selected features
model.feature_importance_

...
```

Classifier

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(criterion="gini",
    splitter="best", max_depth=None, min_samples_split=2,
    min_samples_leaf=1, min_weight_fraction_leaf=0.0,
    max_features=None, random_state=None, max_leaf_nodes=None,
    min_impurity_decrease=0.0, class_weight=None,
    ccp_alpha=0.0, monotonic_cst=None)
# To train the model
model.fit(X_train, y_train)

# Get path of decision making (num_samples, num_nodes)
path = model.decision_path(x)

# Get selected features
model.feature_importance_

...
```

Let’s practice

- On the Iris dataset (on Canvas) try to train both a Logistic Regression and a Decision Tree using scikit-learn LogisticRegression and scikit-learn DecisionTreeClassifier.