EEC1509 - Machine Learning

Lesson #09 Decision Trees

Ivanovitch Silva

November, 2018

Update repository

```
git clone https://github.com/ivanovitchm/EEC1509_MachineLearning.git
```

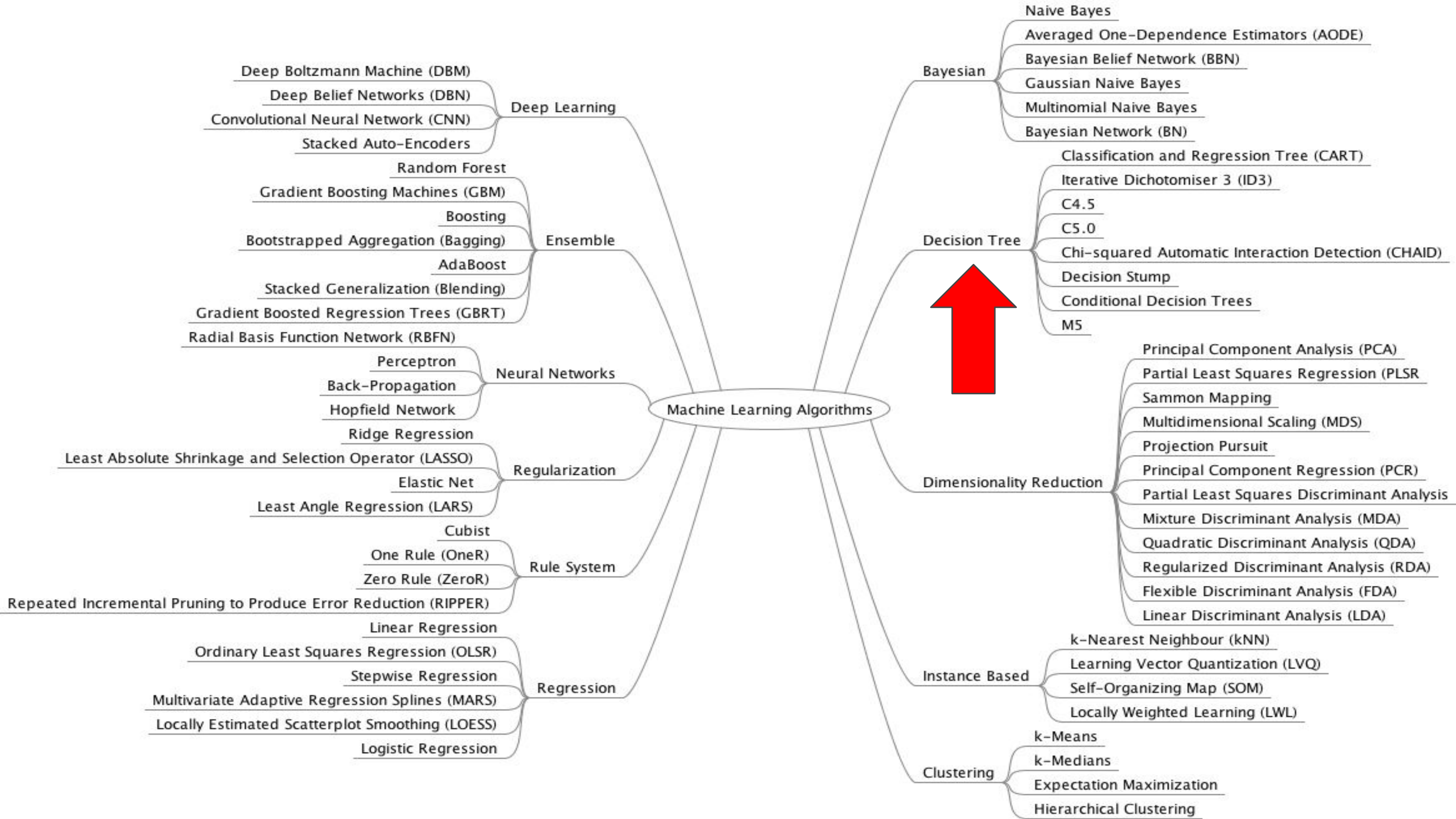
Ou ....

```
git pull
```

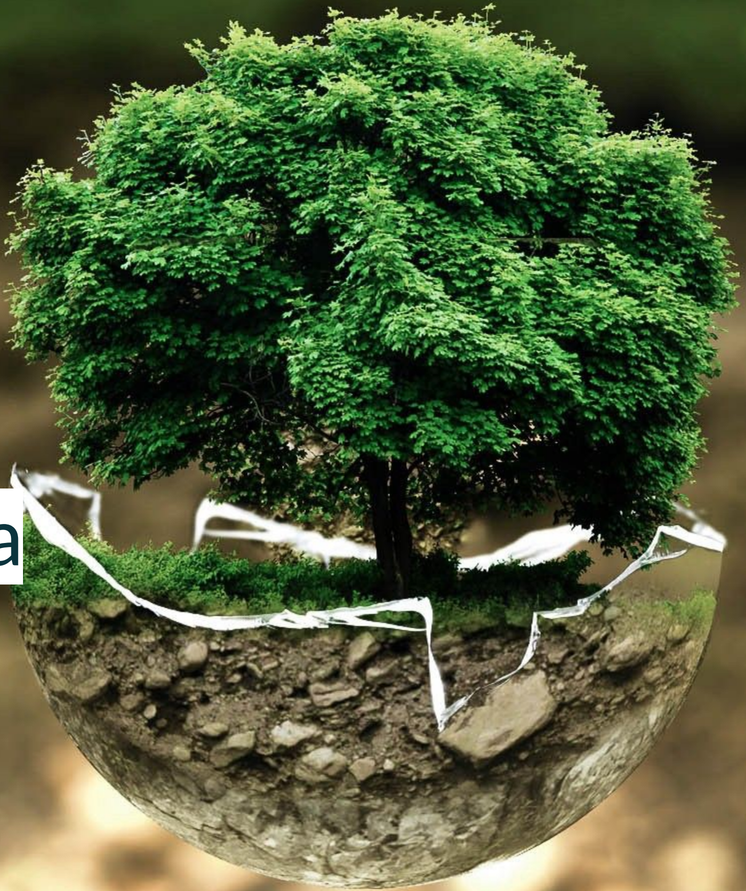
GitHub

Agenda

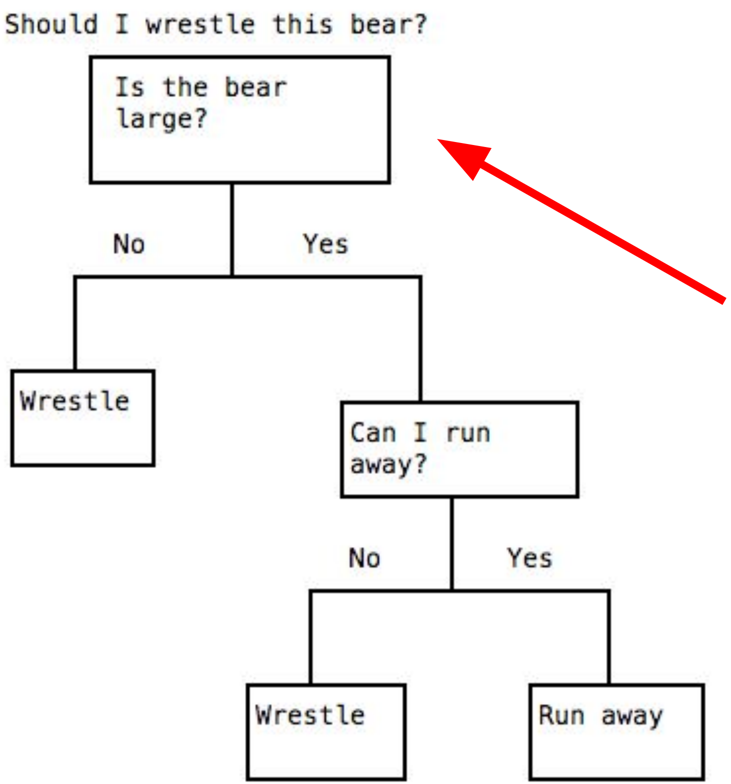
- Introduction to Decision Tree
- Converting categorical variables
- Splitting Data
- Decision Trees as flows of data
- Entropy
- Information gain
- Applying Decision Trees
- Overfitting problem


Machine Learning Algorithms
Deep Learning
Deep Boltzmann Machine (DBM)
Deep Belief Networks (DBN)
Convolutional Neural Network (CNN)
Stacked Auto-Encoders
Ensemble
Random Forest
Gradient Boosting Machines (GBM)
Boosting
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (Blending)
Gradient Boosted Regression Trees (GBRT)
Neural Networks
Radial Basis Function Network (RBFN)
Perceptron
Back-Propagation
Hopfield Network
Regularization
Ridge Regression
Least Absolute Shrinkage and Selection Operator (LASSO)
Elastic Net
Least Angle Regression (LARS)
Rule System
Cubist
One Rule (OneR)
Zero Rule (ZeroR)
Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
Regression
Linear Regression
Ordinary Least Squares Regression (OLSR)
Stepwise Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Logistic Regression
Bayesian
Naive Bayes
Averaged One-Dependence Estimators (AODE)
Bayesian Belief Network (BBN)
Gaussian Naive Bayes
Multinomial Naive Bayes
Bayesian Network (BN)
Decision Tree
Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5
C5.0
Chi-squared Automatic Interaction Detection (CHAID)
Decision Stump
Conditional Decision Trees
M5
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLSR
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Principal Component Regression (PCR)
Partial Least Squares Discriminant Analysis
Mixture Discriminant Analysis (MDA)
Quadratic Discriminant Analysis (QDA)
Regularized Discriminant Analysis (RDA)
Flexible Discriminant Analysis (FDA)
Linear Discriminant Analysis (LDA)
Instance Based
k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Locally Weighted Learning (LWL)
Clustering
k-Means
k-Medians
Expectation Maximization
Hierarchical Clustering

How can an algorithm be represented as a tree?

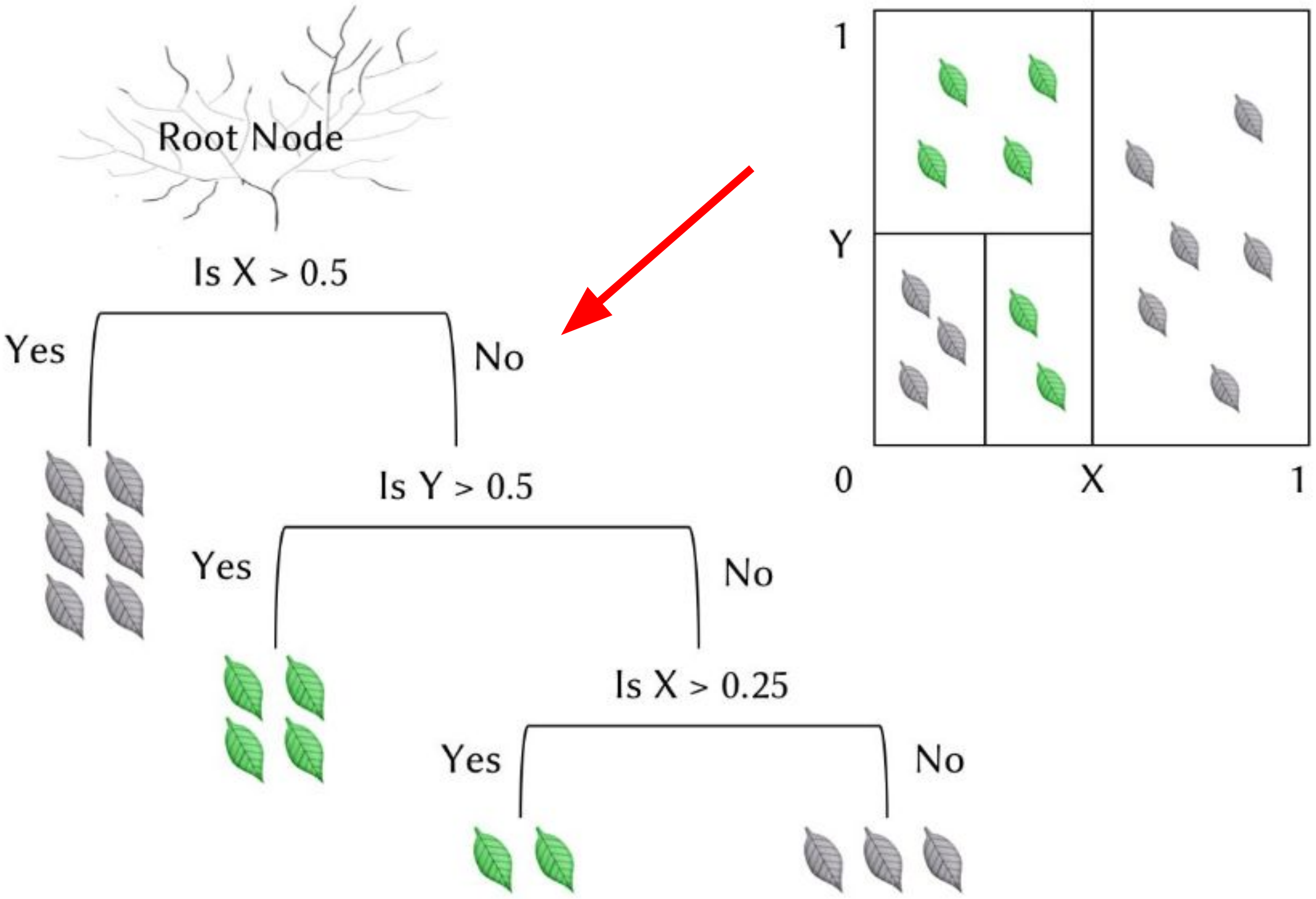


Decision Tree (classification)



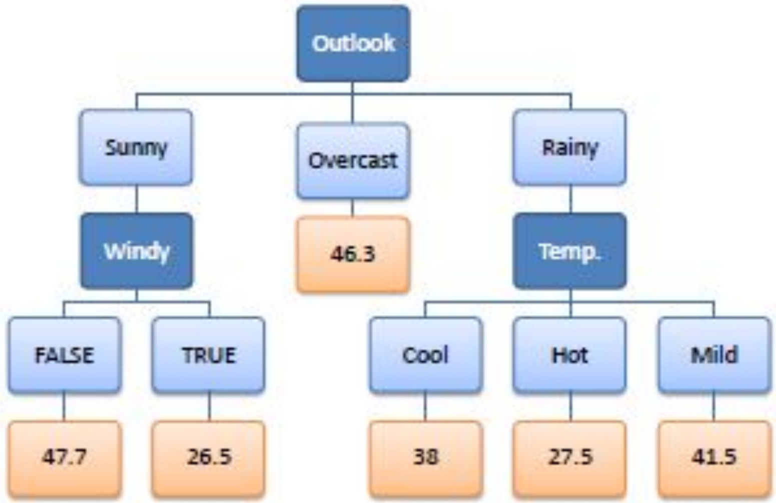
Bear name	Size	Escape possible?	Action
Yogi	Small	No	Wrestle
Winnie	Small	Yes	Wrestle
Baloo	Large	Yes	Run away
Gentle Ben	Large	No	Wrestle

Decision Tree (classification)



Decision Tree (regression)

Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	26
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	62
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



The Dataset

The target column, or what we want to predict, is whether individuals make less than or equal to 50k a year, or more than 50k a year.

US Census 1994

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0

Converting Categorical Variables

```
income.workclass.head()
```

```
0           State-gov
1    Self-emp-not-inc
2             Private
3             Private
4             Private
```

```
# Convert a single column from text categories to numbers
col = pd.Categorical(income["workclass"])
income["workclass"] = col.codes
income.workclass.head()
```

```
0    7
1    6
2    4
3    4
4    4
```

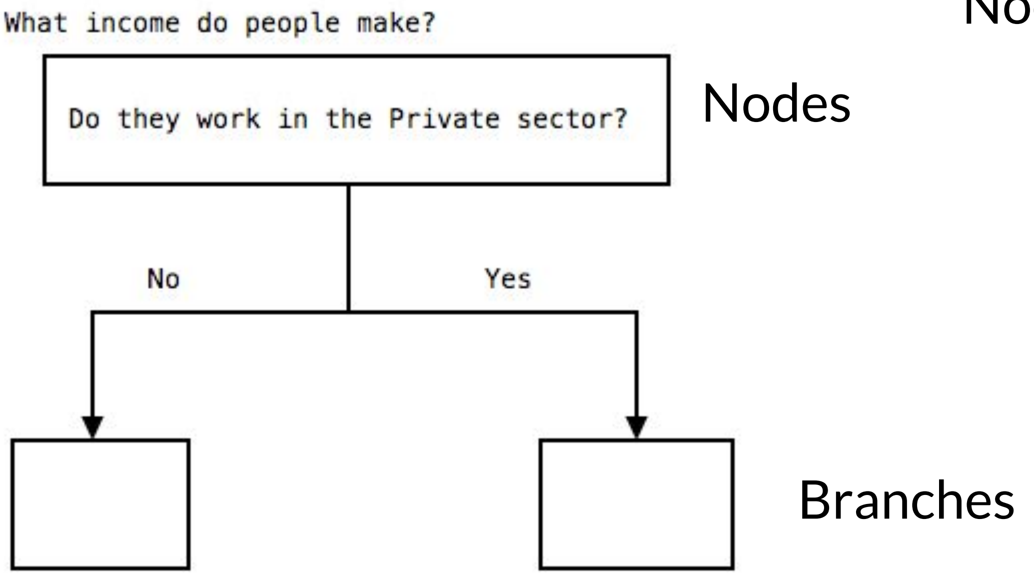
```
col.categories[7]
```

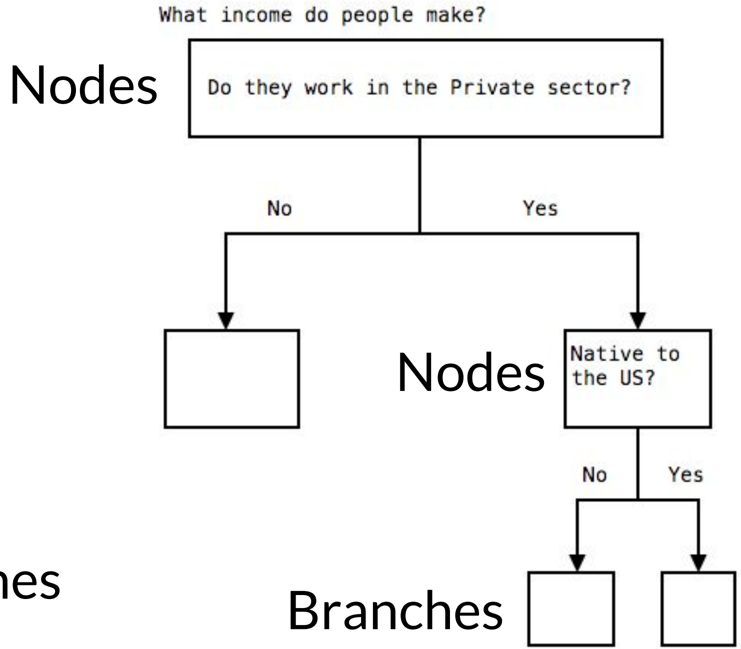
```
' State-gov'
```

```
col.categories
```

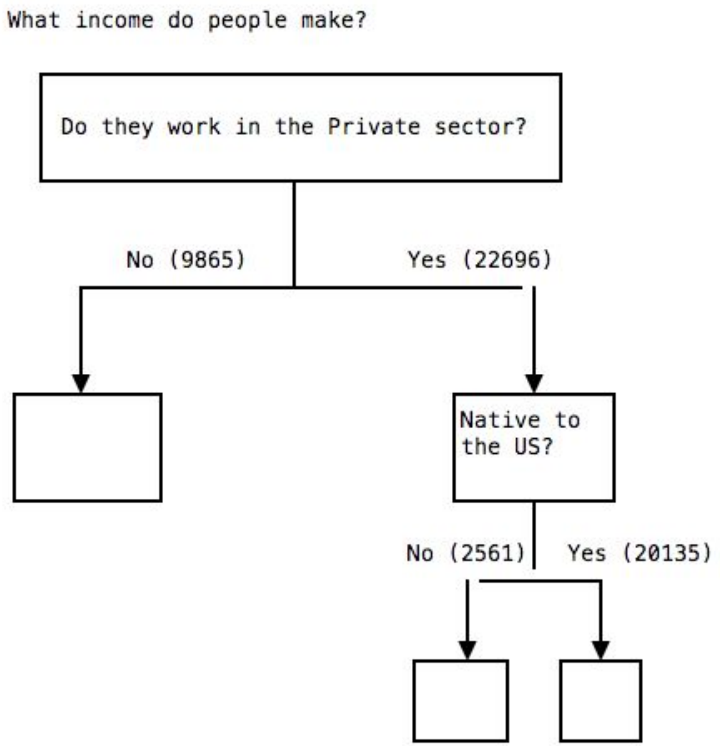
```
Index([' ?', ' Federal-gov', ' Local-gov', ' Never-worked', ' Private',
       ' Self-emp-inc', ' Self-emp-not-inc', ' State-gov', ' Without-pay'],
      dtype='object')
```

Splitting Data





Decision Tree as Flows of Data

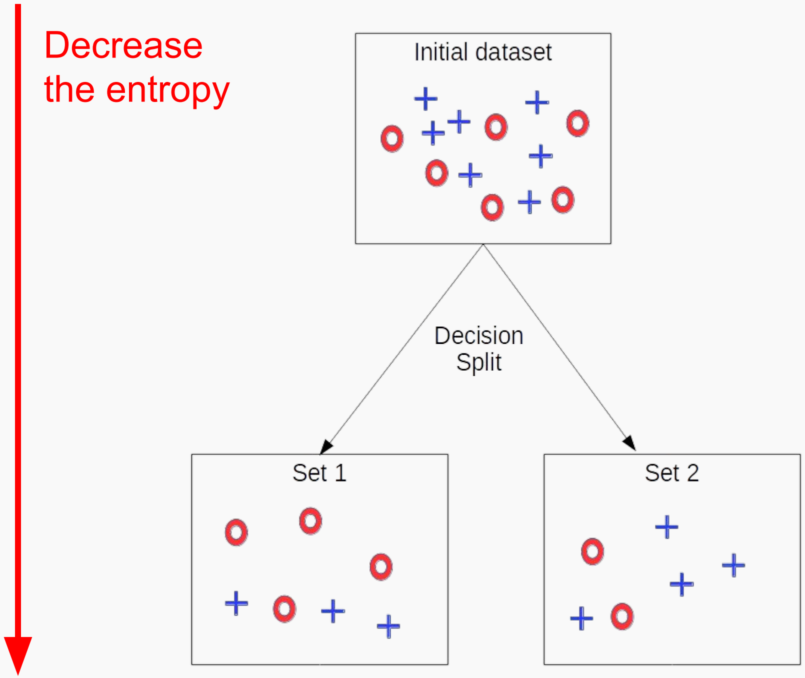


We'll need to continue splitting nodes until we get to a point where all of the rows in a node have the same value for high_income.

Entropy

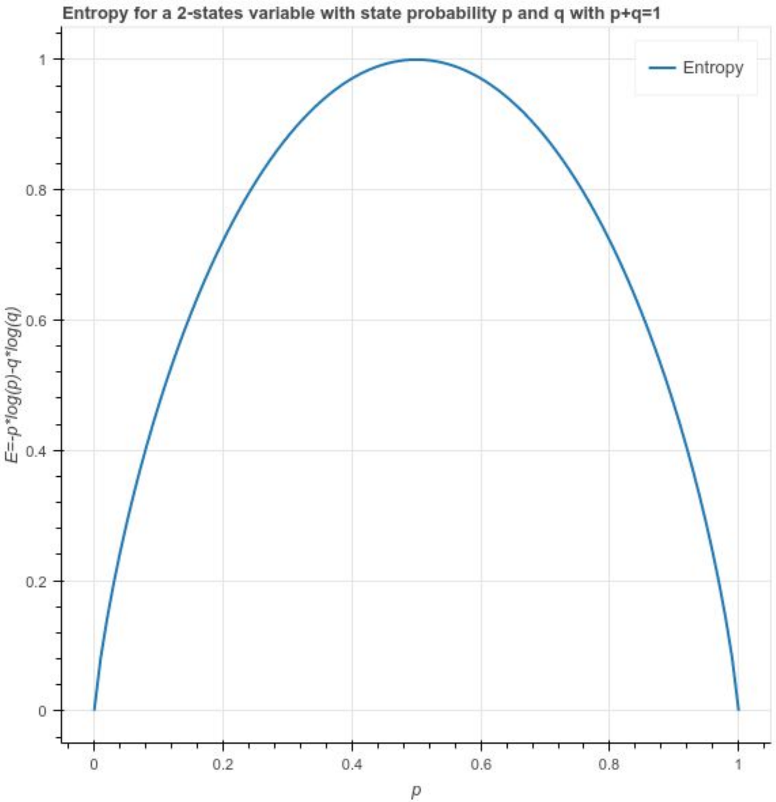
Entropy is an indicator of how messy your data is.

Why Entropy in Decision Trees?



- The goal is to tidy the data.
- You try to separate your data and group the samples together in the classes they belong to.
- You maximize the purity of the groups as much as possible each time you create a new node of the tree
- Of course, at the end of the tree, you want to have a clear answer.

Mathematical definition of entropy



- Suppose a set of N items, these items fall into two categories:
  - Label 1: n items
  - Label 2: m items
- p = n/N
- q =m/N
- p + q = 1
- E = -p log(p) - q log(q)

Generalization

$$E(x) = -\sum_{i=1}^{c} P(x_i) \log_b P(x_i)$$

c = pd.x.unique()

Entropy using the frequency table of one attribute

high_income
1
1
0
0
1

$$E(x) = -\sum_{i=1}^{c} P(x_i)\log_b P(x_i)$$

$$E(high\_income) = -((2/5 * log_2 2/5) + (3/5 * log_2 3/5))$$

$$E(high\_income) = -(-0.5287712379549449 + -0.44217935649972373)$$

$$E(high\_income) = 0.97$$

Entropy using the frequency table of two attributes

age	high_income	split_age
25	1	0
50	1	0
30	0	0
50	0	0
80	1	1

split_age is based on median of age (suppose equal to 50)

$$E(T, X) = \sum_{c \in X} P(c)E(c)$$

$$E(high\_income, split\_age) = \frac{4}{5} E(split\_age, 0) + \frac{1}{5} E(split\_age, 1)$$

$$E(split\_age, 0) = -\left(\frac{1}{2} \times \log_2 \frac{1}{2} + \frac{1}{2} \times \log_2 \frac{1}{2}\right)$$

$$E(split\_age, 1) = -(0 \times \log_2 0 + 1 \times \log_2 1)$$

$$E(high\_income, split\_age) = \frac{4}{5}$$

Information Gain

$$IG(T,X) = E(T) - E(T,X)$$

The information gain is based on the **decrease in entropy after a dataset is split** on an attribute.

Constructing a decision tree is all about finding attribute that returns the **highest information gain** (i.e., the most homogeneous branches).

Applying Decision Tree

sklearn.tree.DecisionTreeClassifier

class sklearn.tree. **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, min_impurity_split=None, class_weight=None, presort=False*) [source]

sklearn.tree.DecisionTreeRegressor

class sklearn.tree. **DecisionTreeRegressor** (*criterion='mse', 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, min_impurity_split=None, presort=False*) ¶ [source]

Applying Decision Tree

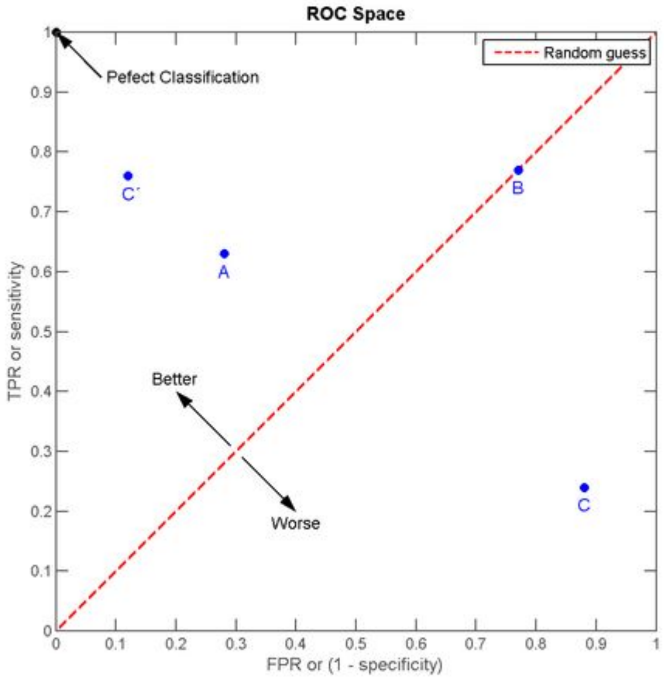
```
from sklearn.tree import DecisionTreeClassifier

# A list of columns to train with
# We've already converted all columns to numeric
columns = ["age", "workclass", "education_num", "marital_status",
           "occupation", "relationship", "race",
           "sex", "hours_per_week", "native_country"]

# Instantiate the classifier
# Set random_state to 1 to make sure the results are consistent
clf = DecisionTreeClassifier(random_state=1)

# fit using features and target
clf.fit(income[columns], income["high_income"])
```

Receiver Operating Characteristic (ROC)



		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error
	Predicted condition negative	**False negative**, Type II error	**True negative**
		True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$

AUC - Area Under Curve

Decision Tree Overfitting

```
from sklearn.metrics import roc_auc_score

clf = DecisionTreeClassifier(random_state=1)
clf.fit(train[columns], train["high_income"])

predictions = clf.predict(test[columns])
error = roc_auc_score(test["high_income"], predictions)
print(error)
```

0.6934656324746192

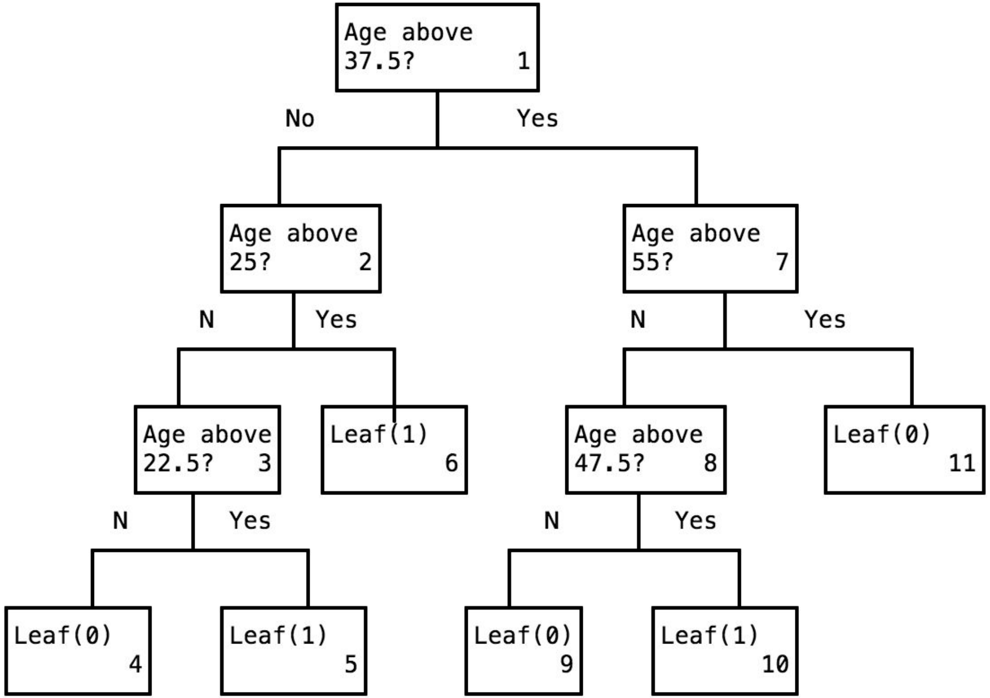
```
predictions = clf.predict(train[columns])
print(roc_auc_score(train["high_income"], predictions))
```

0.9471244501437455

Splitting the data into training and testing sets doesn't prevent overfitting -- it just helps us detect and fix it.

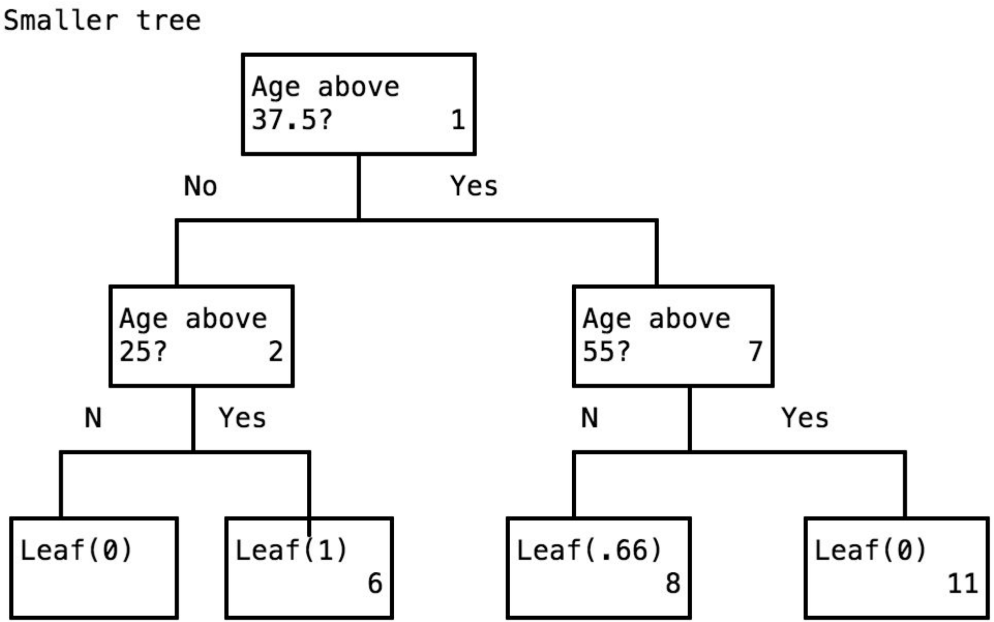
Decision Tree Overfitting

Full tree



- If you're under 22.5 years old, you have a low income
- If you're 22.5 - 37.5, you have a high income
- If you're 37.5 - 47.5, you have a low income
- If you're 47.5 to 55, you have a high income
- Finally, if you're above 55, you have a low income.

Decision Tree Overfitting



This version actually has lower accuracy on our training set, but will generalize to new examples better because it matches reality more closely.

Reducing Overfitting with a Shallower Tree

- **max_depth** - Globally restricts how deep the tree can go
- **min_samples_split** - The minimum number of rows a node should have before it can be split; if this is set to 2, for example, then nodes with 2 rows won't be split, and will become leaves instead
- **min_samples_leaf** - The minimum number of rows a leaf must have
- **min_weight_fraction_leaf** - The fraction of input rows a leaf must have
- **max_leaf_nodes** - The maximum number of total leaves; this will cap the count of leaf nodes as the tree is being built

settings	train AUC	test AUC
default	0.947	0.694
min_samples_split: 13	0.842	0.699

settings	train AUC	test AUC
default (min_samples_split: 2, max_depth: None)	0.947	0.694
min_samples_split: 13	0.842	0.699
min_samples_split: 13, max_depth: 7	0.748	0.743
min_samples_split: 100, max_depth: 2	0.662	0.655

Knowing when to use decision trees

The main advantages of using decision trees is that they're:

- Easy to interpret
- Relatively fast to fit and make predictions
- Able to handle multiple types of data
- Able to pick up nonlinearities in data, and usually fairly accurate

The main disadvantage of using decision trees is their **tendency to overfit.**

Lesson #09 - Decision Trees.ipynb