Supervised Learning

Tree-based Models

Peerapon S.

Machine Learning

Codes & Dataset: https://kmutt.me/cpe-ml

Topics

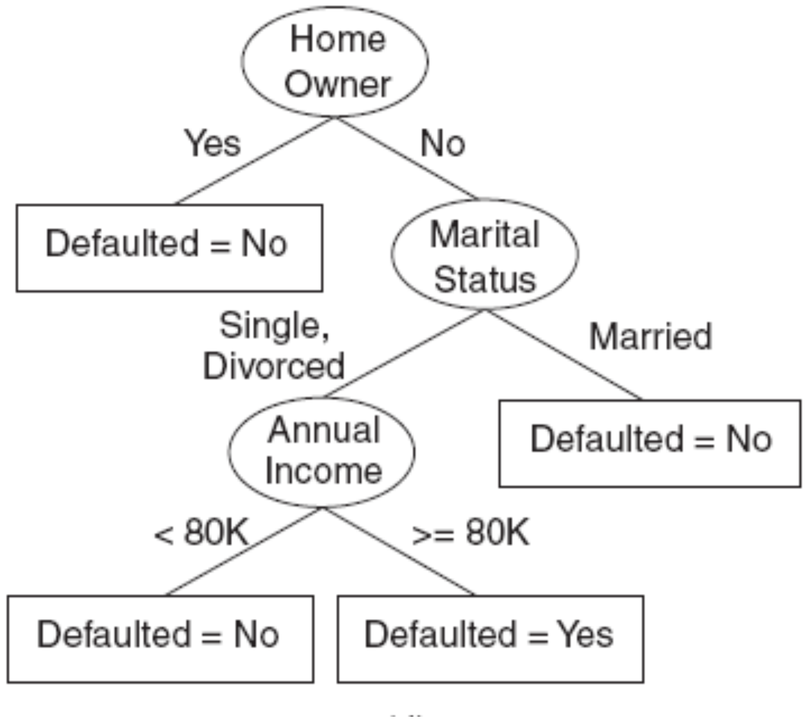
- Classification tree
  - Basic operations
  - Impurity measures
  - Attribute selection
- Python implementation
- Regression tree

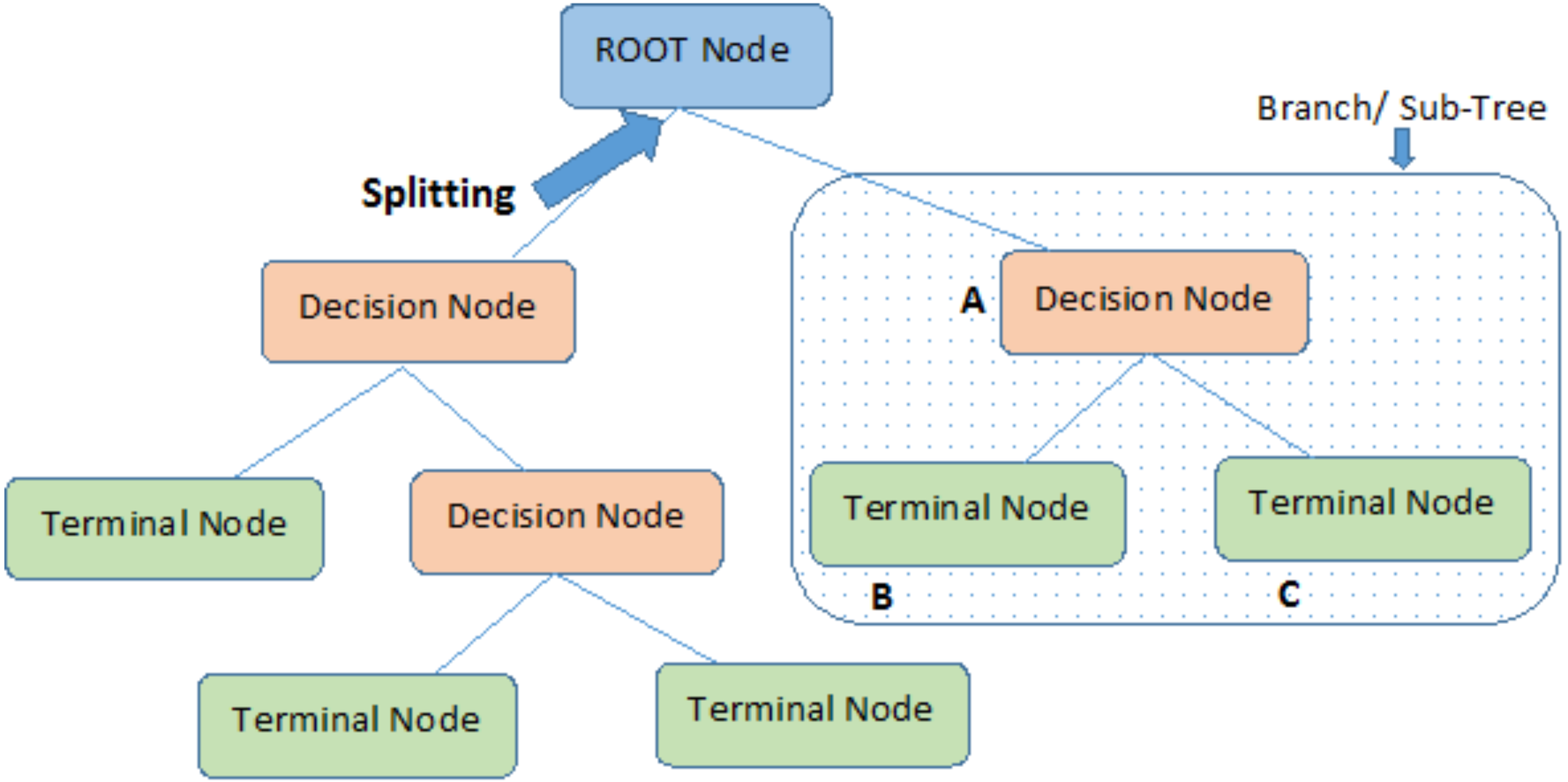
Decision Tree Concept

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Model parameters = Features to split dataset at each node

Model fitting = Select appropriate features at each node to get "pure" data subsets at the bottom of the tree.





Note:- A is parent node of B and C.

1. How should the records be split at each node ?
2. When to stop splitting ?

Measures of Impurity

- Data set is more pure (less information) as more records belong to a certain class.
- Less information = More impurity
- Common measures
  - Entropy
  - Gini index
  - Classification error
  - χ^2 test statistic

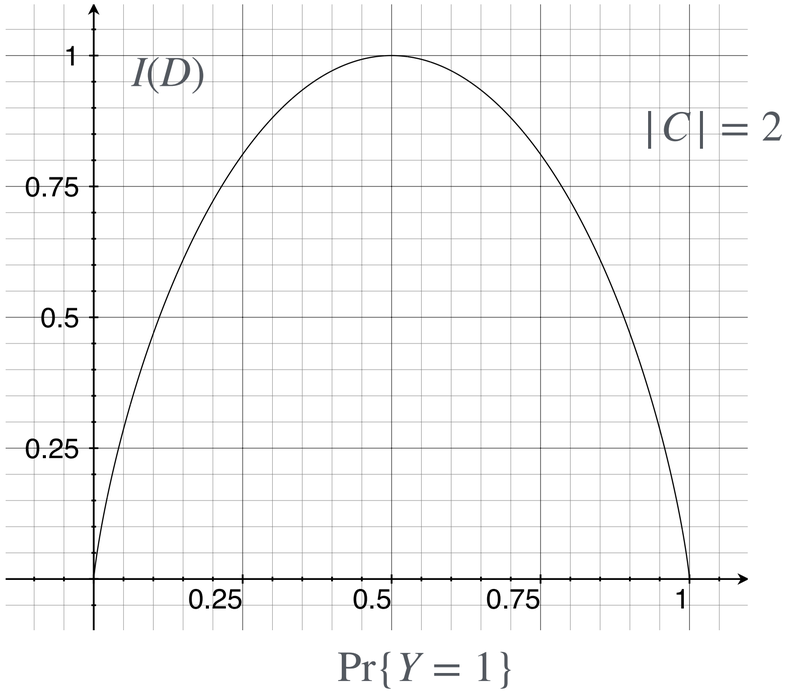
Entropy

- Consider a data set D with a set of class labels C and p_c being the proportion of records in class $c \in C$.
- **Entropy** of the data set is defined by

$$I(D) = -\sum_{\forall c \in \mathcal{C}} p_c \log_2 p_c, \quad p_c = \mathbb{P}\{Y = c\}$$

  - Quantify the amount of "information" in the data.
  - Characterize the “unpredictability” of a random variable

- Ex: Binary class output ($|C| = 2$)
  - $I = 0$: Purest data set (lowest entropy)
  - $I = 1$: Most impure data set (1/2 each)



Ex: Employee Enrollment in Onsite Educational Prog.

- **Attributes**
  - X_1 = Age: {≤ 30, 31 to 40, > 40}
  - X_2 = Income: {Low, Medium, High}
  - X_3 = Job Satisfaction: {Yes, No}
  - X_4 = Desire: {Fair, Excellent}
- **Output**: Enrolls {Yes, No}
- Want to predict which employees would enroll in an on-site educational program.

☐ Find the entropy enrollment data set.

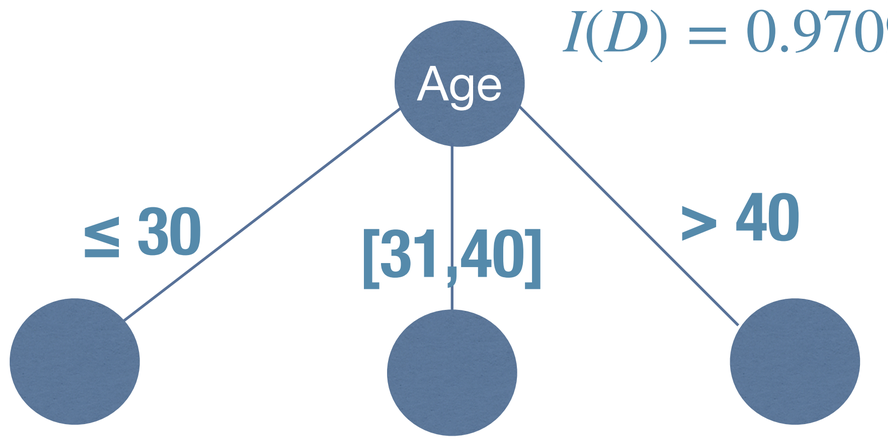
	A	B	C	D	E
1	Age	Income	Jobsatisfaction	Desire	Enrolls
2	<=30	High	No	Fair	No
3	<=30	High	No	Excellent	No
4	31 to 40	High	No	Fair	Yes
5	>40	Medium	No	Fair	Yes
6	>40	Low	Yes	Fair	Yes
7	>40	Low	Yes	Excellent	No
8	31 to 40	Low	Yes	Excellent	Yes
9	<=30	Medium	No	Fair	No
10	<=30	Low	Yes	Fair	Yes
11	>40	Medium	Yes	Fair	Yes
12	<=30	Medium	Yes	Excellent	Yes
13	31 to 40	Medium	No	Excellent	Yes
14	31 to 40	High	Yes	Fair	Yes
15	>40	Medium	No	Excellent	No

- Find the entropy enrollment data with Age ≤ 30.

	A	B	C	D	E
1	Age	Income	Jobsatisfaction	Desire	Enrolls
2	<=30	High	No	Fair	No
3	<=30	High	No	Excellent	No
4	31 to 40	High	No	Fair	Yes
5	>40	Medium	No	Fair	Yes
6	>40	Low	Yes	Fair	Yes
7	>40	Low	Yes	Excellent	No
8	31 to 40	Low	Yes	Excellent	Yes
9	<=30	Medium	No	Fair	No
10	<=30	Low	Yes	Fair	Yes
11	>40	Medium	Yes	Fair	Yes
12	<=30	Medium	Yes	Excellent	Yes
13	31 to 40	Medium	No	Excellent	Yes
14	31 to 40	High	Yes	Fair	Yes
15	>40	Medium	No	Excellent	No

Impurity of Subsets from Node Splitting

- Suppose attribute X is used to split parent data set D with N records into k subsets (nodes) $v_1, v_2, \ldots, v_k$ with sizes $N(v_1), N(v_2), \ldots, N(v_k)$
- Let $I(v_i)$ be the entropy of data in subset v_i



$I(D) = 0.9709$

$N(v_1) = 5$

$I(v_1) = 0.9709$ $I(v_2) = 0$ $I(v_3) = 0.9709$

Selecting Splitting Attribute

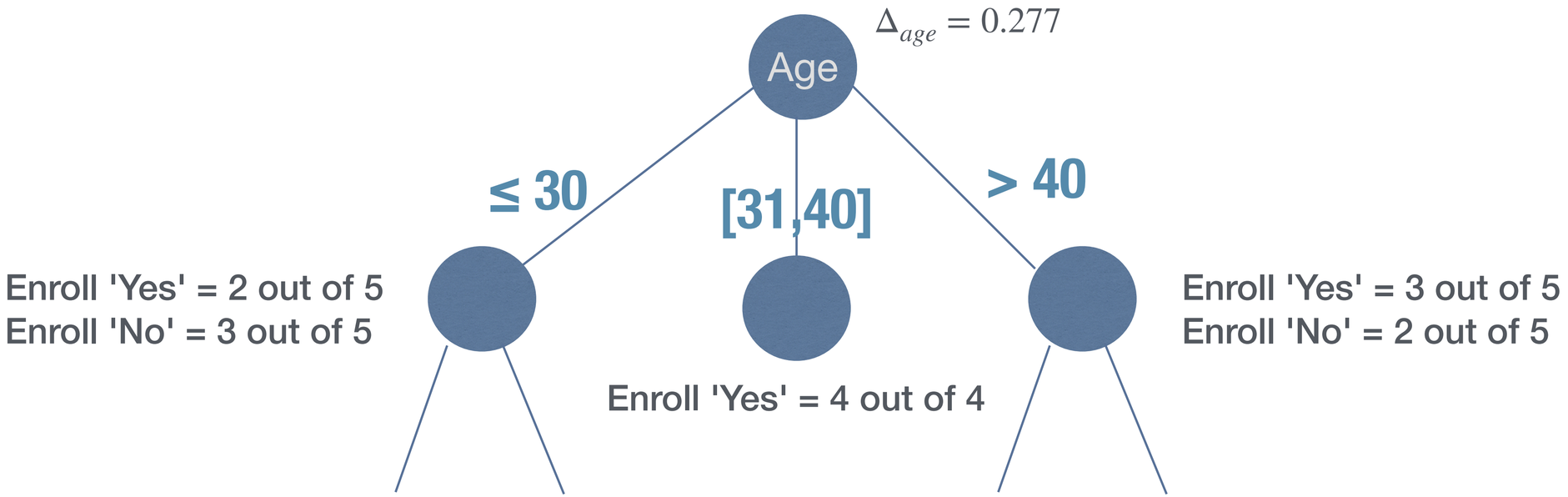
- Then, the improvement of purity (information gain) using X to split data D is

$$\Delta = I(D) - \sum_{i=1}^{k} \frac{N(v_i)}{N} I(v_i)$$

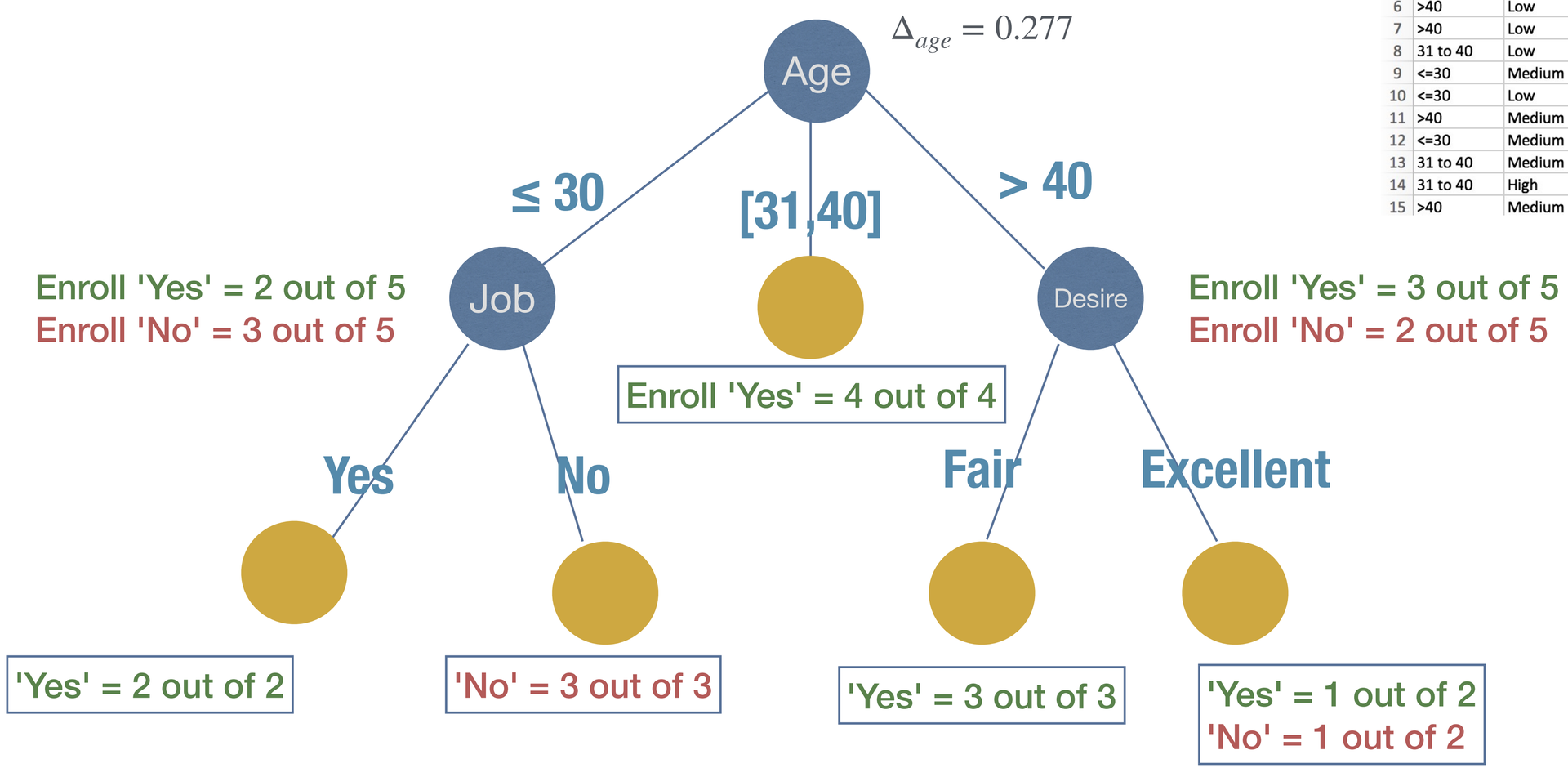
$$= I(D) - I(D, X)$$

- Ex: For the attribute Age,
  - $I(D, Age)$ = {(5/14)*0.9709 + (4/14)*0 + (5/14)*0.9709} = 0.6935
  - Δ_{Age} = 0.94 - 0.6935 = 0.2465
- Compute the gain of each attribute and pick the one with lowest $I(D, X)$ for splitting (**most informative attribute**)

First-Level Splitting



	A	B	C	D	E
1	Age	Income	Jobsatisfaction	Desire	Enrolls
2	<=30	High	No	Fair	No
3	<=30	High	No	Excellent	No
4	31 to 40	High	No	Fair	Yes
5	>40	Medium	No	Fair	Yes
6	>40	Low	Yes	Fair	Yes
7	>40	Low	Yes	Excellent	No
8	31 to 40	Low	Yes	Excellent	Yes
9	<=30	Medium	No	Fair	No
10	<=30	Low	Yes	Fair	Yes
11	>40	Medium	Yes	Fair	Yes
12	<=30	Medium	Yes	Excellent	Yes
13	31 to 40	Medium	No	Excellent	Yes
14	31 to 40	High	Yes	Fair	Yes
15	>40	Medium	No	Excellent	No



Generic Decision Tree Algorithm

- Starting from the whole data set as the root node, recursively split each node by using an attribute that yields the highest gain.
- Termination criteria:
  - Node has the impurity below a pre-specified threshold (most of the records belong to the same class), or
  - Node size becomes too small, or
  - Insufficient impurity (gain) improvement after the last splitting, or
  - Tree depth exceeds a pre-specified threshold.

Other Impurity Measures

- Gini index

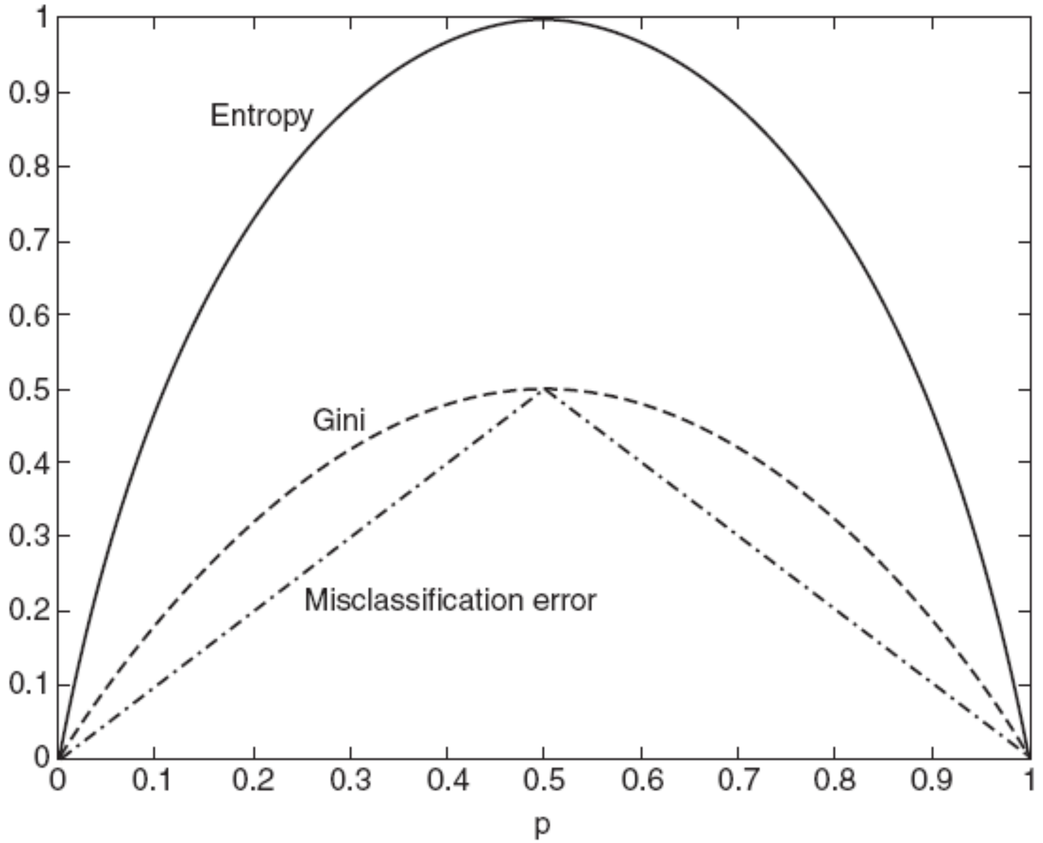
$$I(D) = 1 - \sum_{\forall c \in \mathcal{C}} p_c^2$$

  - $I = 0$: Purest data set
  - $I = 0.5$: Most impure data set (1/2 each)

- Classification error

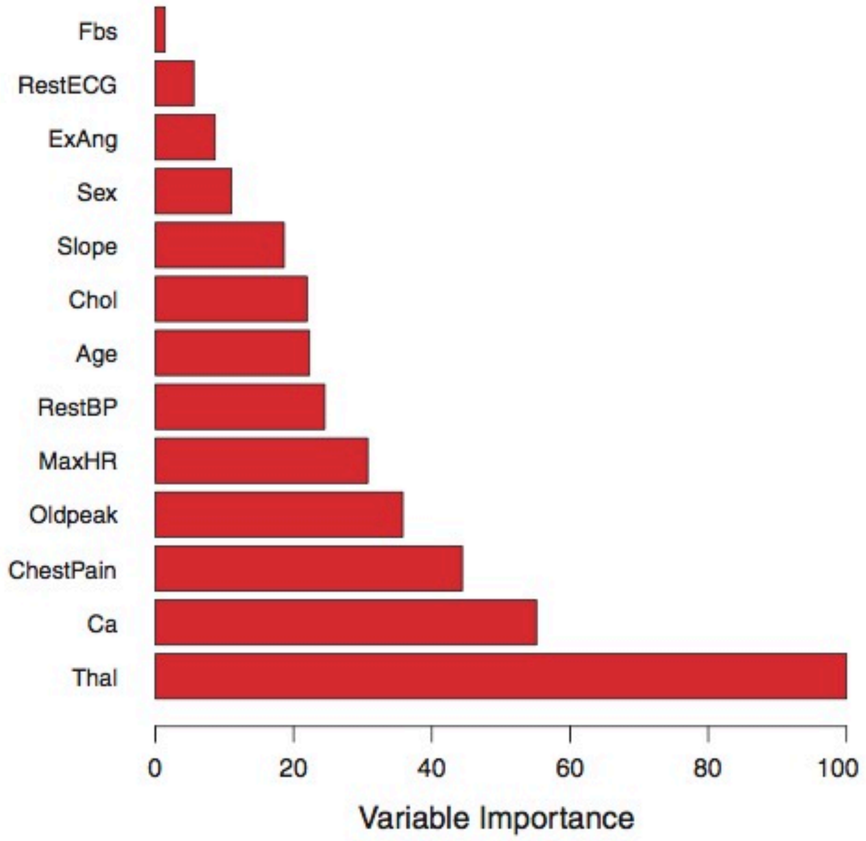
$$I(D) = 1 - \max_{c \in \mathcal{C}} p_c$$

  - $I = 0$: Purest data set
  - $I = 0.5$: Most impure data set (1/2 each)

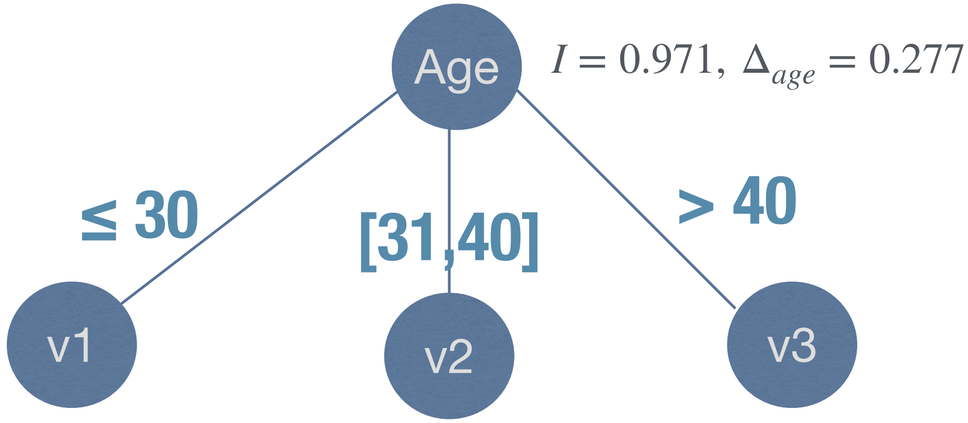


Feature Importance Measure

- For each tree, add up the gain Δ of a given feature when splitting.

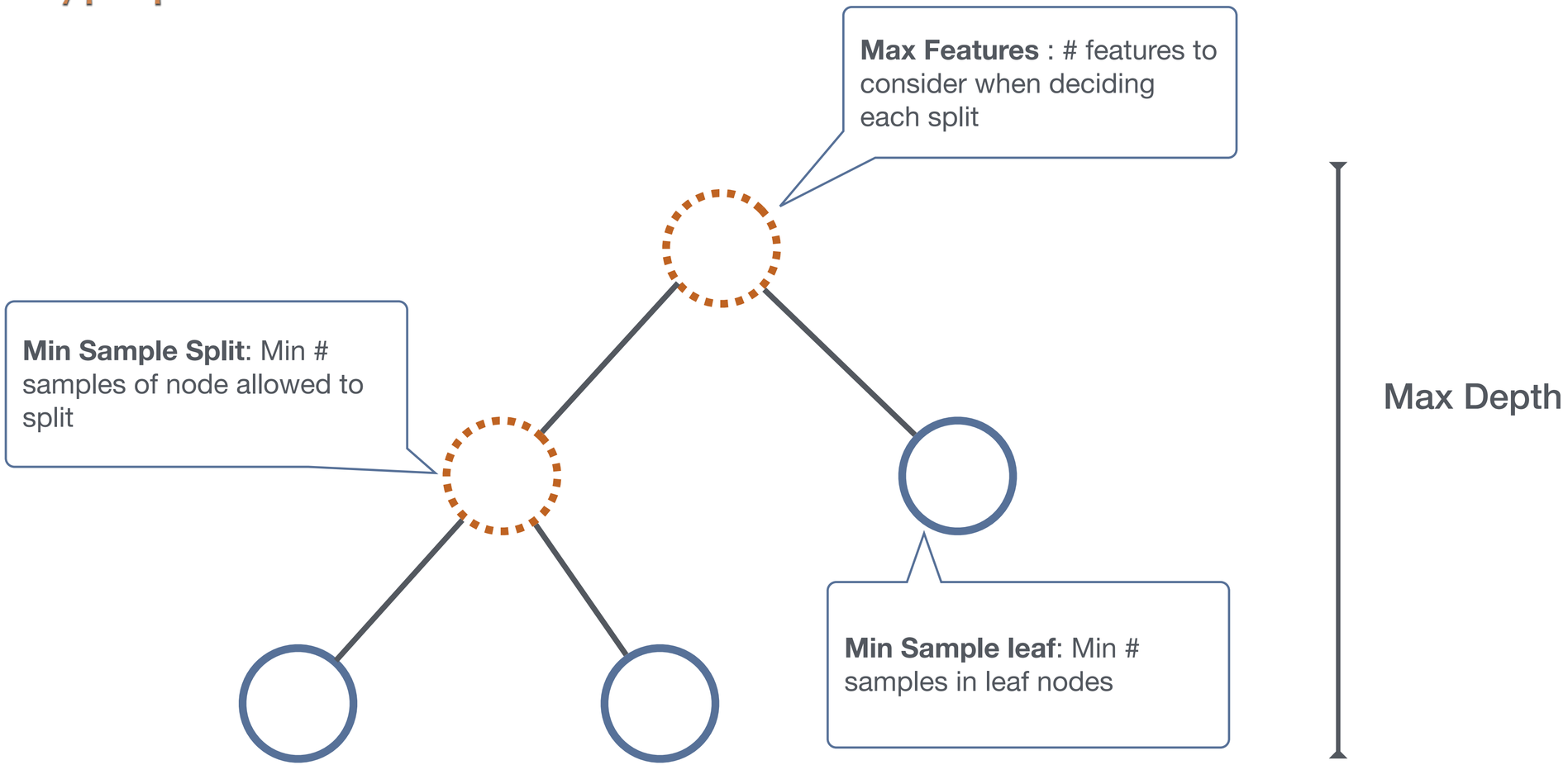


$$\Delta = I(\text{parent}) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j)$$

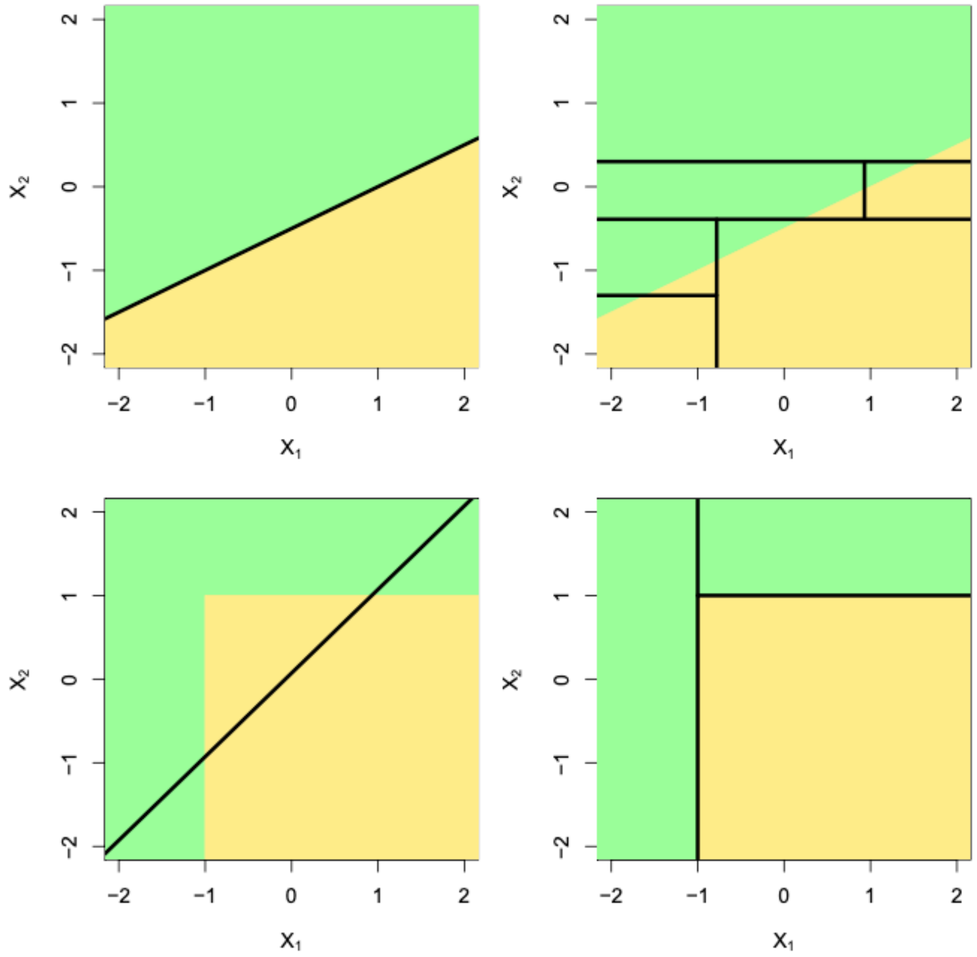


[Jame23] Figure 8.9

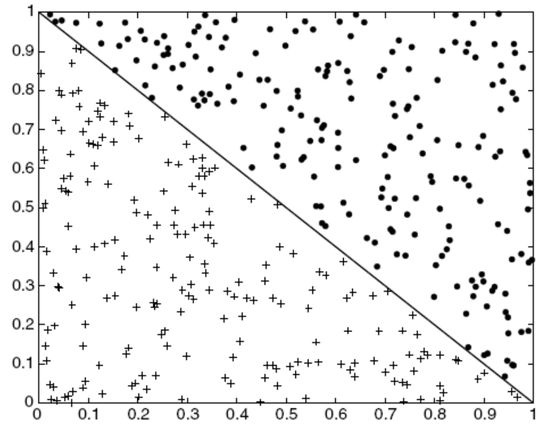
Decision Tree Hyperparameters



Decision Boundaries



[Jame23] Figure 8.7



Decision Tree Implementation using Python

- ☐ Load and explore data
  - Basic information (dimension, column meanings, etc.)
  - Missing values, Correlations
- ☐ Data preparation and preprocessing
  - Drop unused or redundant columns
  - Label grouping
  - Data encoding
  - Splitting data
- ☐ Model fitting
- ☐ Performance evaluation and model tunning
- ☐ Visualizing results

Load and Explore Data

- Basic information

```
# Look at the first few rows
enroll_df.head()
```

	Age	Income	JobSat	Desire	Target
0	<=30	High	No	Fair	No
1	<=30	High	No	Excellent	No
2	31 to 40	High	No	Fair	Yes
3	>40	Medium	No	Fair	Yes
4	>40	Low	Yes	Fair	Yes

```
# Inspect basic information (# rows, data types, missing values, etc.)
enroll_df.info()
enroll_df.describe(include='all')
```

```
RangeIndex: 14 entries, 0 to 13
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Age     14 non-null     object
 1   Income  14 non-null     object
 2   JobSat  14 non-null     object
 3   Desire  14 non-null     object
 4   Target  14 non-null     object
```

	Age	Income	JobSat	Desire	Target
count	14	14	14	14	14
unique	3	3	2	2	2
top	<=30	Medium	No	Fair	Yes
freq	5	6	7	8	9

```
# Look at unique values of categorical columns
enroll_df.Age.unique()
enroll_df.JobSat.unique()
enroll_df.Desire.unique()
```

```
array(['<=30', '31 to 40', '>40'], dtype=object)
array(['No', 'Yes'], dtype=object)
array(['Fair', 'Excellent'], dtype=object)
```

Data from sheet 'ENROLL' in supervised-learning.xlsx

Data Preprocessing

- One-hot encoding

```
from sklearn.preprocessing import OneHotEncoder

df = enroll_df.copy()

enc_onehot = OneHotEncoder(handle_unknown='ignore',
                           drop='if_binary', dtype=int)
onehot_columns = ['JobSat','Desire']

onehot_data = enc_onehot.fit_transform(df[onehot_columns])

onehot_df = pd.DataFrame(onehot_data.toarray(),
                         columns=enc_onehot.get_feature_names_out(onehot_columns))

df = df.drop(onehot_columns, axis=1);
df = pd.concat([df, onehot_df], axis=1)
df
```

	Age	Income	Target	JobSat_Yes	Desire_Fair
0	<=30	High	No	0	1
1	<=30	High	No	0	0
2	31 to 40	High	Yes	0	1
3	>40	Medium	Yes	0	1
4	>40	Low	Yes	1	1
5	>40	Low	No	1	0
6	31 to 40	Low	Yes	1	0
7	<=30	Medium	No	0	1
8	<=30	Low	Yes	1	1
9	>40	Medium	Yes	1	1
10	<=30	Medium	Yes	1	0
11	31 to 40	Medium	Yes	0	0
12	31 to 40	High	Yes	1	1
13	>40	Medium	No	0	0

☐ Ordinal encoding

```
from sklearn.preprocessing import OrdinalEncoder
import numpy as np

ord_columns = ['Age', 'Income']

Age_cat = ['⩽30', '31 to 40', '>40']
Income_cat = ['Low', 'Medium', 'High']

enc_ord = OrdinalEncoder(categories=[Age_cat, Income_cat],
                         dtype=np.int64)

df[ord_columns] = enc_ord.fit_transform(df[ord_columns])
df
```

```
X = df.drop('Target', axis=1)
Y = df[['Target']]
```

	Age	Income	Target	JobSat_Yes	Desire_Fair
0	0	2	No	0	1
1	0	2	No	0	0
2	1	2	Yes	0	1
3	2	1	Yes	0	1
4	2	0	Yes	1	1
5	2	0	No	1	0
6	1	0	Yes	1	0
7	0	1	No	0	1
8	0	0	Yes	1	1
9	2	1	Yes	1	1
10	0	1	Yes	1	0
11	1	1	Yes	0	0
12	1	2	Yes	1	1
13	2	1	No	0	0

Model Fitting

- Decision tree in scikit-learn implements ***Classification and Regression Tree (CART)*** algorithm.
  - Binary split using Gini index by default.
  - Random or best split

```
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, class_weight=None,
ccp_alpha=0.0, monotonic_cst=None)                                    [source]
```

Version 1.6.1

```python
from sklearn.tree import DecisionTreeClassifier

dt_clf = DecisionTreeClassifier()
dt_clf.fit(X_train, y_train)
```

```
# See the features used in the model and their label
dt_clf.feature_names_in_
enc_ord.categories_
enc_onehot.categories_
```

```
array(['Age', 'Income', 'JobSat_Yes', 'Desire_Fair'], dtype=object)
[array(['<=30', '31 to 40', '>40'], dtype=object),
 array(['Low', 'Medium', 'High'], dtype=object)]
[array(['No', 'Yes'], dtype=object),
 array(['Excellent', 'Fair'], dtype=object)]
```

```
# Encode the input {Age > 40, Income='Medium', Job_Sat='No', Desire='Excellent'}
Xnew = enc_ord.transform([['>40', 'Medium']])
Xnew = np.hstack((Xnew, enc_onehot.transform([['No', 'Excellent']]).toarray()))
Xnew
```

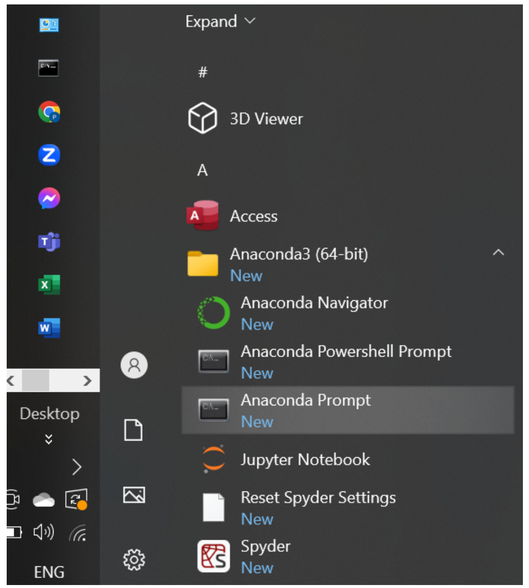
```
array([[2, 1, 0, 0]])
```

```
# Make the prediction to a new instance
dt_clf.predict(Xnew)
dt_clf.predict_proba(Xnew)
```

```
array(['No'], dtype=object)
array([[1., 0.]])
```

Visualizing Rules and Trees

- At the anaconda prompt, install the following packages:




```
conda install python-graphviz

pip install pydotplus
```

```python
from sklearn import tree

text_representation = tree.export_text(dt_clf,
                                       feature_names=dt_clf.feature_names_in_.tolist())
print(text_representation)
```

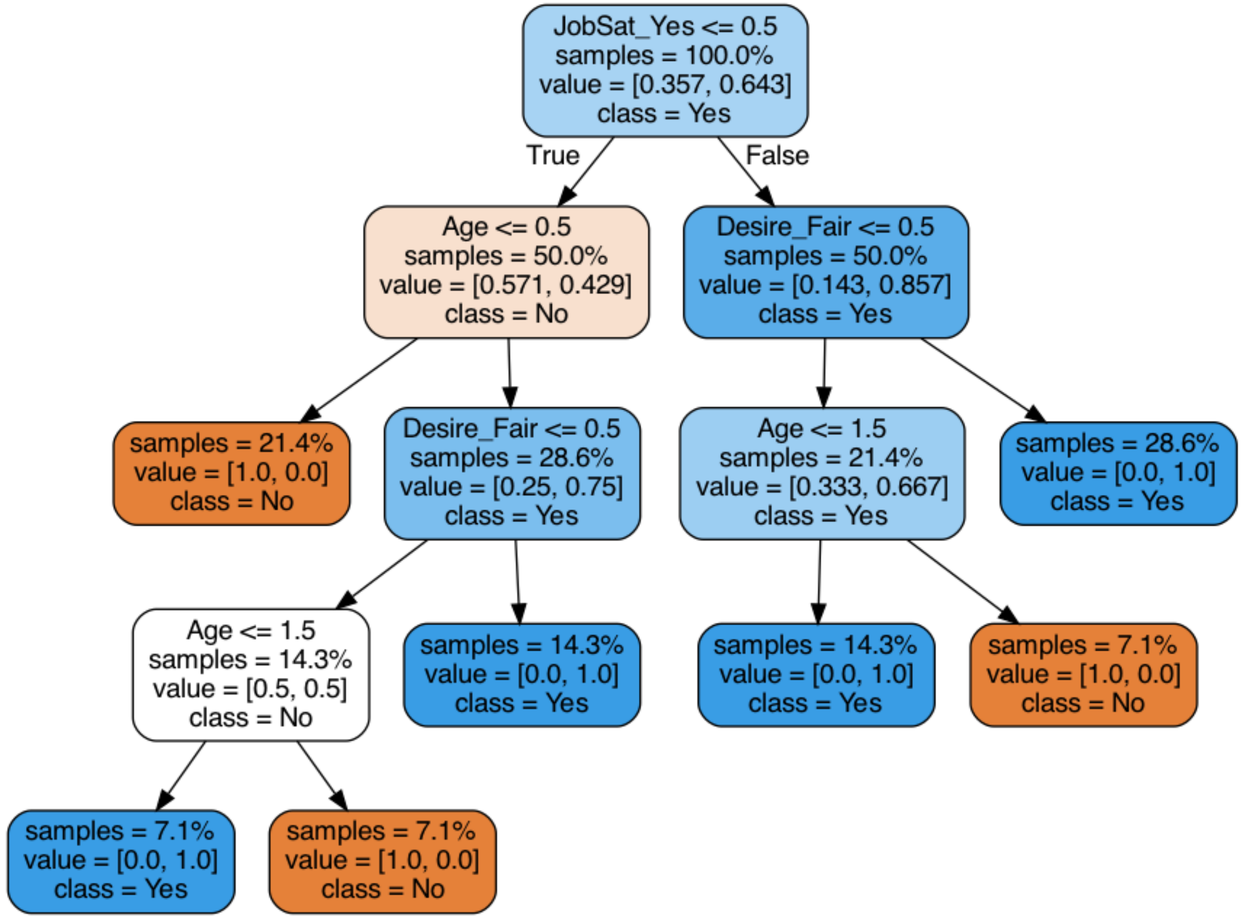
```
|--- JobSat_Yes <= 0.50
|   |--- Age <= 0.50
|   |   |--- class: No
|   |--- Age >  0.50
|   |   |--- Desire_Fair <= 0.50
|   |   |   |--- Age <= 1.50
|   |   |   |   |--- class: Yes
|   |   |   |--- Age >  1.50
|   |   |   |   |--- class: No
|   |   |--- Desire_Fair >  0.50
|   |   |   |--- class: Yes
|--- JobSat_Yes >  0.50
|   |--- Desire_Fair <= 0.50
|   |   |--- Age <= 1.50
|   |   |   |--- class: Yes
|   |   |--- Age >  1.50
|   |   |   |--- class: No
|   |--- Desire_Fair >  0.50
|   |   |--- class: Yes
```

```
import graphviz
from sklearn import tree

dot_data = tree.export_graphviz(dt_clf, out_file=None,
                                feature_names=X.columns,
                                class_names=enroll_df['Target'].unique(), impurity=False,
                                filled=True, rounded=True, proportion=True)
# Draw and show graph
graph = graphviz.Source(dot_data)
graph.render('enroll', format='png', view=True)
```

Need target label as string to plot.
Integer label won't work.

```
|--- JobSat_Yes <= 0.50
|   |--- Age <= 0.50
|   |   |--- class: No
|   |--- Age >  0.50
|   |   |--- Desire_Fair <= 0.50
|   |   |   |--- Age <= 1.50
|   |   |   |   |--- class: Yes
|   |   |   |--- Age >  1.50
|   |   |   |   |--- class: No
|   |   |--- Desire_Fair >  0.50
|   |   |   |--- class: Yes
|--- JobSat_Yes >  0.50
|   |--- Desire_Fair <= 0.50
|   |   |--- Age <= 1.50
|   |   |   |--- class: Yes
|   |   |--- Age >  1.50
|   |   |   |--- class: No
|   |--- Desire_Fair >  0.50
|   |   |--- class: Yes
```



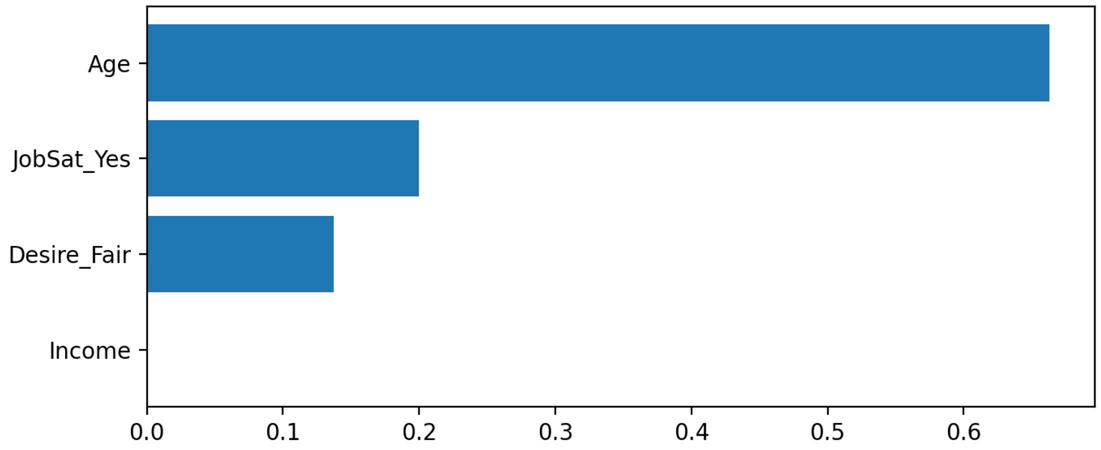
Feature Importance

```
feature_list = pd.DataFrame({'feature':X.columns,
                             'value':dt_clf.feature_importances_})
feature_list_sorted = feature_list.sort_values('value')
feature_list_sorted
```

	feature	value
1	Income	0.000000
3	Desire_Fair	0.137037
2	JobSat_Yes	0.200000
0	Age	0.662963

```
import matplotlib.pyplot as plt

plt.figure(figsize=(7,3))
plt.barh(range(0,len(feature_list_sorted.index)),
         feature_list_sorted.value,
         tick_label=feature_list_sorted.feature)
plt.tight_layout();
```



Does the root node have the largest gain ?

Your Turn: Loan Risk Prediction using Decision Tree

From the Loan train dataset, drop the columns `Id`, `Profession`, `CITY`, and `STATE`.

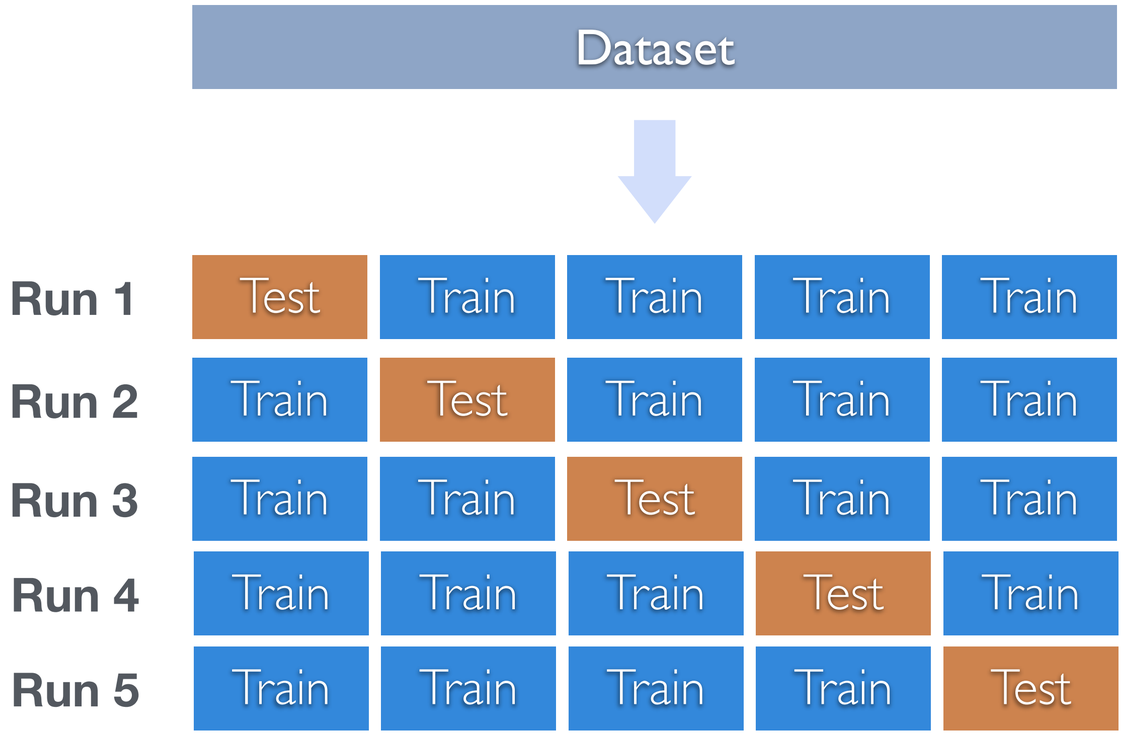
With `Risk_flag` as the target variable, fit the decision tree model with default model options, evaluate the performance on the train data, and determine the feature importance

Apply the fitted model to the test dataset and evaluate the performance.

Train dataset from file data/loans/Trainset.csv
Test dataset from file data/loans/Testset.csv

Model Evaluation: K-Fold Cross-Validation (CV)

- Basic hold-out method fails if the train and test sets are not statistically identical.
- In K-Fold CV, all instances are used so that the evaluation result is less biased.
- Ok if model fitting is not too computationally intensive.



Average of 5 test errors as a measure of generalization performance

K-Fold Cross Validation in Sklearn

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_validate

scoring_metrics = ['accuracy','precision_macro','recall_macro', 'f1_macro']
scores = cross_validate(DecisionTreeClassifier(), X_train, y_train,
                        scoring=scoring_metrics, cv=5, return_train_score=False)
scores_df = pd.DataFrame(scores)
scores_df
scores_df.mean()
```

If the target class is string or multiclass,
use 'precision_macro', 'recall_macro', 'f1_macro'.

	fit_time	score_time	test_accuracy	test_precision	test_recall
0	0.022441	0.008939	0.905782	0.687500	0.647059
1	0.015810	0.006936	0.890792	0.602410	0.735294
2	0.015521	0.007266	0.899358	0.636364	0.720588
3	0.014588	0.007181	0.905579	0.671429	0.691176
4	0.017157	0.007003	0.905579	0.662162	0.720588

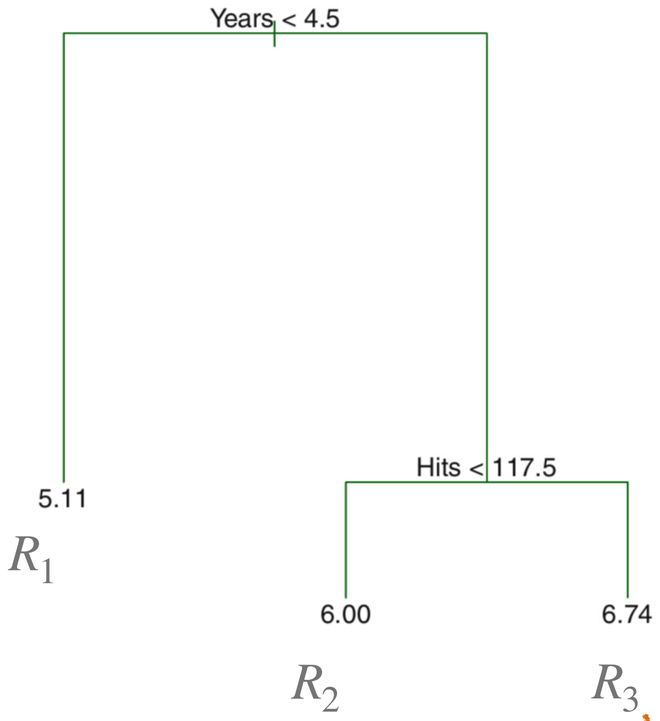
Hitter Dataset

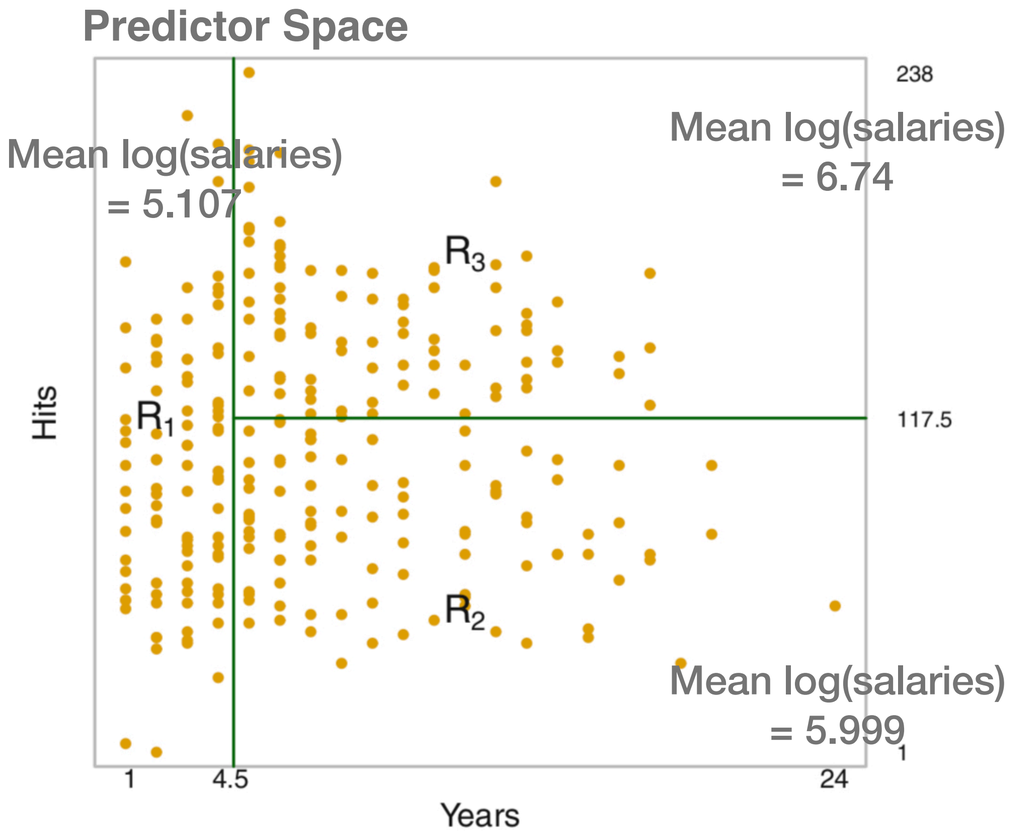
Response
= Natural log of
Yearly salary in 1K

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary	NewLeague
-Andy Allanson	293	66	1	30	29	14	1	293	66	1	30	29	14	A	E	446	33	20	NA	A
-Alan Ashby	315	81	7	24	38	39	14	3449	835	69	321	414	375	N	W	632	43	10	475.000	N
-Alvin Davis	479	130	18	66	72	76	3	1624	457	63	224	266	263	A	W	880	82	14	480.000	A
-Andre Dawson	496	141	20	65	78	37	11	5628	1575	225	828	838	354	N	E	200	11	3	500.000	N
-Andres Galarraga	321	87	10	39	42	30	2	396	101	12	48	46	33	N	E	805	40	4	91.500	N
-Alfredo Griffin	594	169	4	74	51	35	11	4408	1133	19	501	336	194	A	W	282	421	25	750.000	A
-Al Newman	185	37	1	23	8	21	2	214	42	1	30	9	24	N	E	76	127	7	70.000	A
-Argenis Salazar	298	73	0	24	24	7	3	509	108	0	41	37	12	A	W	121	283	9	100.000	A
-Andres Thomas	323	81	6	26	32	8	2	341	86	6	32	34	8	N	W	143	290	19	75.000	N
-Andre Thornton	401	92	17	49	66	65	13	5206	1332	253	784	890	866	A	E	0	0	0	1100.000	A
-Alan Trammell	574	159	21	107	75	59	10	4631	1300	90	702	504	488	A	E	238	445	22	517.143	A
-Alex Trevino	202	53	4	31	26	27	9	1876	467	15	192	186	161	N	W	304	45	11	512.500	N
-Andy VanSlyke	418	113	13	48	61	47	4	1512	392	41	205	204	203	N	E	211	11	7	550.000	N
-Alan Wiggins	239	60	0	30	11	22	6	1941	510	4	309	103	207	A	E	121	151	6	700.000	A
-Bill Almon	196	43	7	29	27	30	13	3231	825	36	376	290	238	N	E	80	45	8	240.000	N
-Billy Beane	183	39	3	20	15	11	3	201	42	3	20	16	11	A	W	118	0	0	NA	A
-Buddy Bell	568	158	20	89	75	73	15	8068	2273	177	1045	993	732	N	W	105	290	10	775.000	N
-Buddy Biancalana	190	46	2	24	8	15	5	479	102	5	65	23	39	A	W	102	177	16	175.000	A
-Bruce Bochte	407	104	6	57	43	65	12	5233	1478	100	643	658	653	A	W	912	88	9	NA	A
-Bruce Bochy	127	32	8	16	22	14	8	727	180	24	67	82	56	N	W	202	22	2	135.000	N
-Barry Bonds	413	92	16	72	48	65	1	413	92	16	72	48	65	N	E	280	9	5	100.000	N
-Bobby Bonilla	426	109	3	55	43	62	1	426	109	3	55	43	62	A	W	361	22	2	115.000	N
-Bob Boone	22	10	1	4	2	1	6	84	26	2	9	9	3	A	W	812	84	11	NA	A
-Bob Brenly	472	116	16	60	62	74	6	1924	489	67	242	251	240	N	W	518	55	3	600.000	N
-Bill Buckner	629	168	18	73	102	40	18	8424	2464	164	1008	1072	402	A	E	1067	157	14	776.667	A
-Brett Butler	587	163	4	92	51	70	6	2695	747	17	442	198	317	A	E	434	9	3	765.000	A

322 records

Regression Tree fitted to Data





Two internal (splitting) nodes

Three terminal/leaves nodes: R_1, R_2, R_3

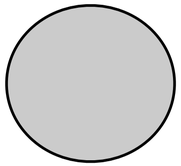
Prediction

$= \$1000 \times e^{6.74} = \$845{,}346$

Minimize $RSS = \sum_{j=1}^{3} \frac{|R_j|}{n} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$

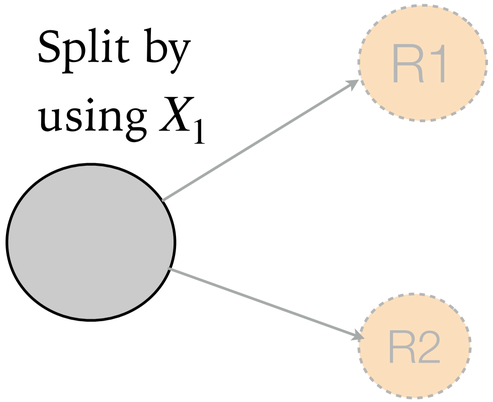
How to Construct Regions: Recursive Binary Splitting

Total $RSS = \sum_{\forall i} (y_i - \bar{y})^2$



Whole data set

Split by using X_1



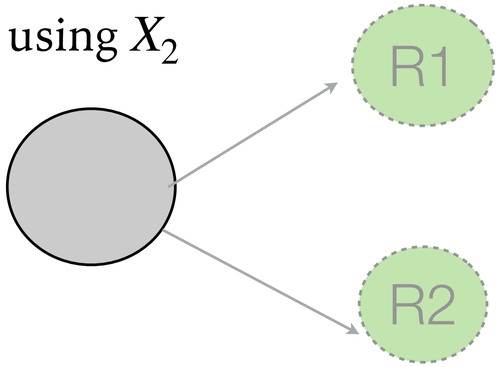
$X_1 < c_1$

$X_1 \geq c_1$

Determine cut point that yields smallest RSS

New $RSS = \frac{|R_1|}{|R|} \sum_{i \in R_1} (y_i - \hat{y}_{R_1})^2 + \frac{|R_2|}{|R|} \sum_{i \in R_2} (y_i - \hat{y}_{R_2})^2$

Split by using X_2

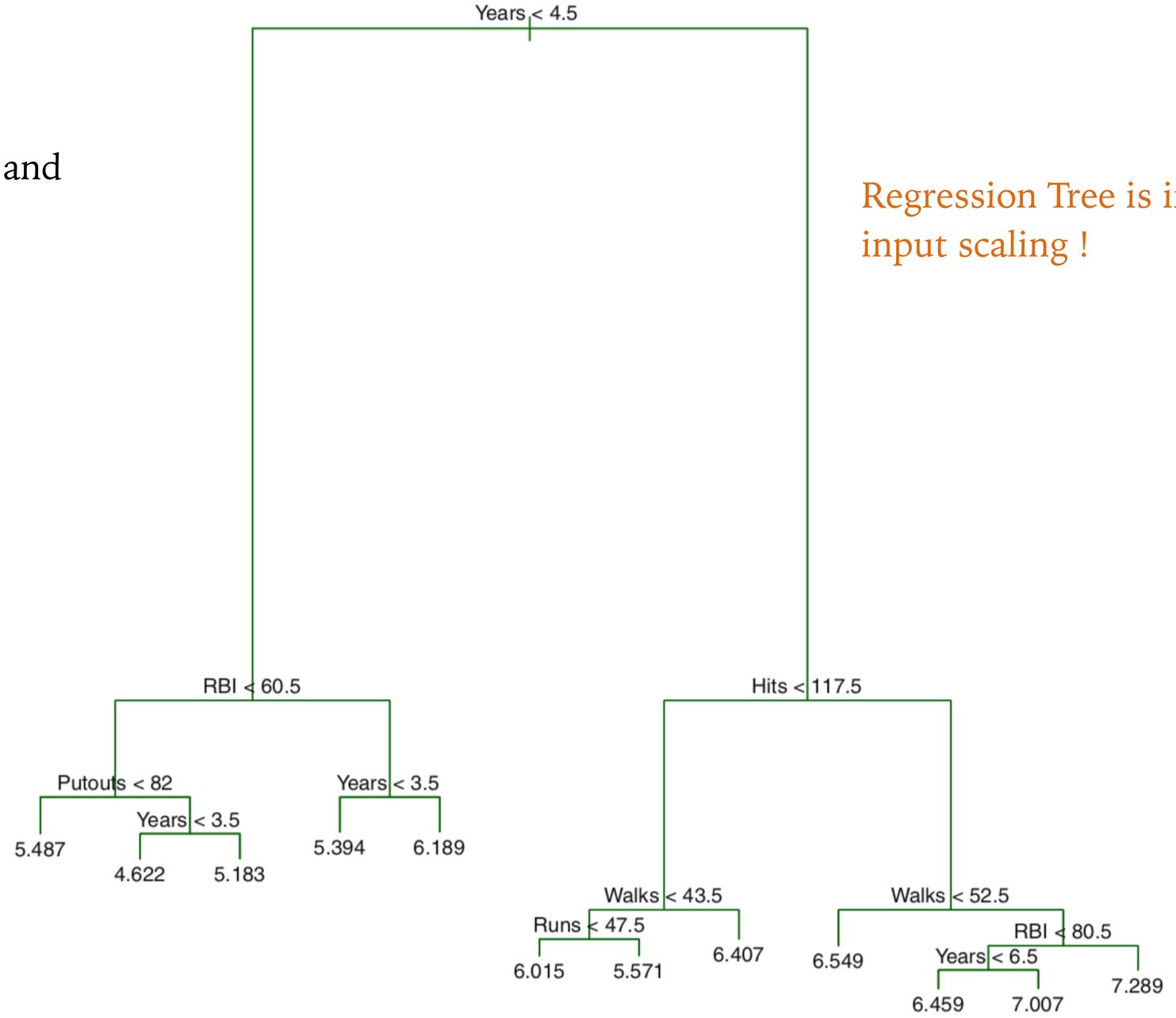


$X_2 < c_2$

$X_2 \geq c_2$

New $RSS = \frac{|R_1|}{|R|} \sum_{i \in R_1} (y_i - \hat{y}_{R_1})^2 + \frac{|R_2|}{|R|} \sum_{i \in R_2} (y_i - \hat{y}_{R_2})^2$

Tree with more features and more depth



Regression Tree is insensitive to input scaling !

Cost Complexity (Weakest Link) Pruning

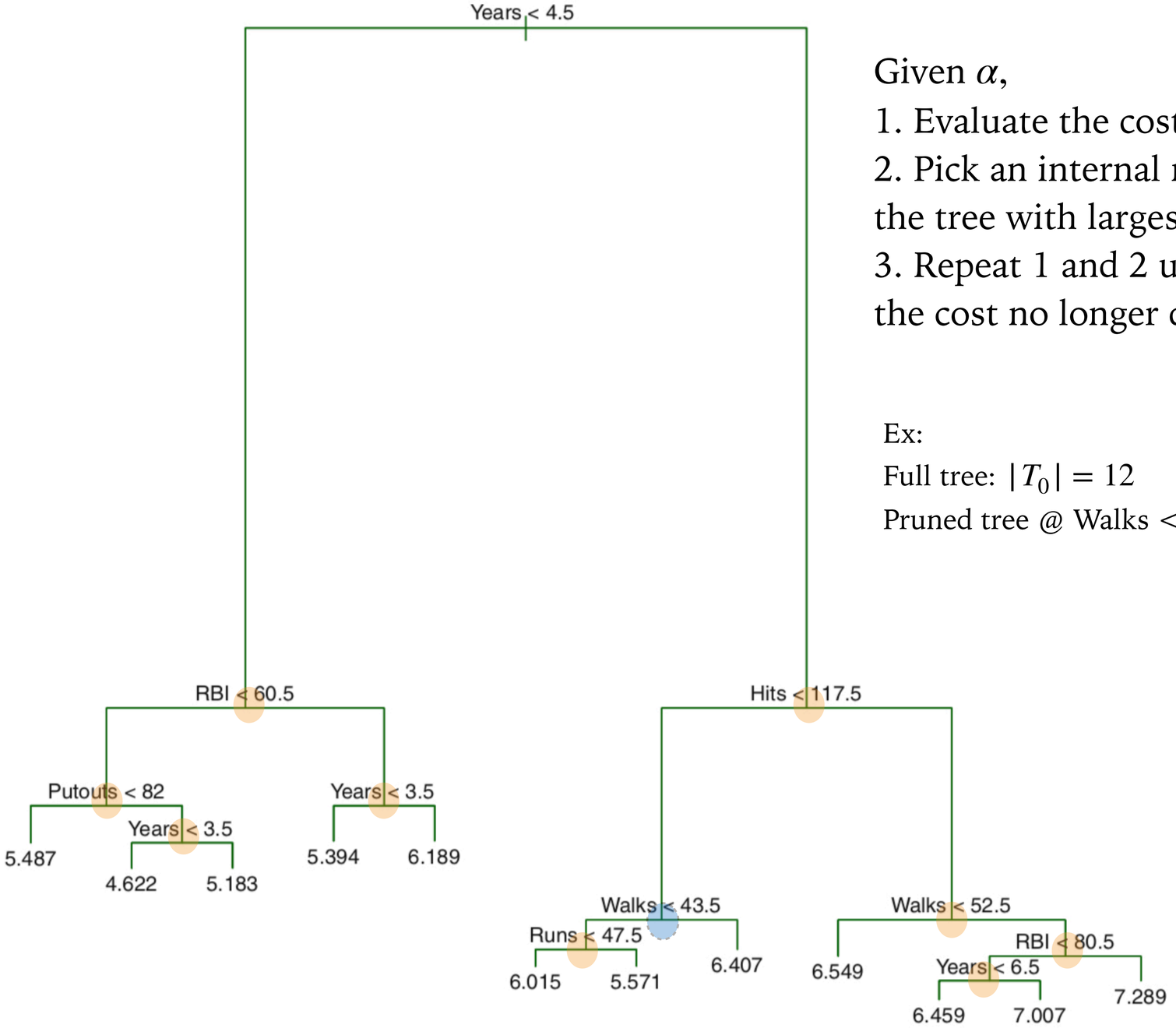
- Grow very large tree T_0 and prune it to select a subtree with lower (test) MSE.
- Given a tuning parameter α, determine a subtree $T \subset T_0$ to minimize the cost function:

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

Region of m^{th} terminal node

Trade-off subtree complexity and its fit to training data

Terminal nodes



Given α,

1. Evaluate the cost function.
2. Pick an internal node to prune the tree with largest cost reduction.
3. Repeat 1 and 2 until the cost no longer decreases.

Ex:

Full tree: $|T_0| = 12$

Pruned tree @ Walks < 43.5: $|T| = 10$

Model Fitting

- Regression tree in scikit-learn implements Classification and Regression Tree (CART) algorithm.
  - Binary split using mean squared error by default
  - Random or best split

```
class sklearn.tree.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)                                                [source]
```

Version 1.6.1

```python
from sklearn.tree import DecisionTreeRegressor

dt_reg = DecisionTreeRegressor()
dt_reg.fit(X_train, y_train)

# Predict results
dt_reg.predict(X_train[0:2])
dt_reg.predict_proba(X_train[0:2])
```

Summary of DT

- ☐ Relatively intuitive to interpret (by Human)
- ☐ Easily handle qualitative variables (no dummy var)
- ☐ Allow ranking of feature importance (feature selection)
- ☐ Non-parametric classifier - No prior assumptions of probability distributions on classes and attributes.
- ☐ Scikit-learn implementation not suitable for lots of categorical variables
- ☐ Tend to overfit (Low bias, High variance) and sensitive to data set.