Step 1: Calculating Shannon Entropy

from math import log

import operator

# Calculating Shannon Entropy

def calculate\_entropy(data):

label\_counts = {}

for feature\_data in data:

Laber = feature\_data [-1] The last line is laber

if laber not in label\_counts.keys():

label\_counts[laber] = 0

label\_counts[laber] += 1

count = len(data)

entropy = 0.0

for key in label\_counts:

prob = float(label\_counts[key]) / count

entropy -= prob \* log(prob, 2)

return entropy

Step 2. A Method of Calculating Information Gain of a Feature

# Calculating the Information Gain of a Feature

# Index: Which column of data corresponds to the feature to calculate the information gain

# Shannon Entropy of Data

def calculate\_relative\_entropy(data, index, entropy):

Feat\_list= [number [index] for number in data] # Gets all the values under a feature (a column)

uniqual\_vals = set(feat\_list)

new\_entropy = 0

for value in uniqual\_vals:

sub\_data = split\_data(data, index, value)

prob = len(sub\_data) / float(len(data))

New\_entropy += prob \* calculate\_entropy (sub\_data) Summation of Shannon Entropy for each subset

Relative\_entropy = entropy - new\_entropy# Calculates information gain

return relative\_entropy

Step 3. Selecting features with maximum information gain

# feature selection for maximum information gain

def choose\_max\_relative\_entropy(data):

num\_feature = len(data[0]) - 1

Base\_entropy = calculate\_entropy (data)# Shannon Entropy

best\_infor\_gain = 0

best\_feature = -1

for i in range(num\_feature):

info\_gain=calculate\_relative\_entropy(data, i, base\_entropy)

# Maximum Information Gain

if (info\_gain > best\_infor\_gain):

best\_infor\_gain = info\_gain

best\_feature = i

return best\_feature

Step 4. Building Decision Trees

def create\_decision\_tree(data, labels):

class\_list=[example[-1] for example in data]

# Categories are the same, stop dividing

if class\_list.count(class\_list[-1]) == len(class\_list):

return class\_list[-1]

# The category with the largest number of returns when judging whether all the features have been traversed

if len(data[0]) == 1:

return most\_class(class\_list)

# Selection of Classified Characteristic Attributes Based on Maximum Information Gain

best\_feat = choose\_max\_relative\_entropy(data)

Best\_feat\_lable = labels [best\_feat] label for this feature

Decision\_tree= {best\_feat\_lable: {}} Dictionary for building trees

Del (labels [best\_feat]) Delete the label from the list of labels

feat\_values = [example[best\_feat] for example in data]

unique\_values = set(feat\_values)

for value in unique\_values:

sub\_lables=labels[:]

# Construct a subset of data and recurse it

decision\_tree[best\_feat\_lable][value] = create\_decision\_tree(split\_data(data, best\_feat, value), sub\_lables)

return decision\_tree

Two tool approaches are used in the process of building a decision tree:

# Returns the largest number of categories when traversing all features

def most\_class(classList):

class\_count={}

for vote in classList:

if vote not in class\_count.keys():class\_count[vote]=0

class\_count[vote]+=1

sorted\_class\_count=sorted(class\_count.items,key=operator.itemgetter(1),reversed=True)

return sorted\_class\_count[0][0]

# Tool function inputs three variables (data set, feature, classification value) to return a subset without partition feature

def split\_data(data, axis, value):

ret\_data=[]

for feat\_vec in data:

if feat\_vec[axis]==value :

reduce\_feat\_vec=feat\_vec[:axis]

reduce\_feat\_vec.extend(feat\_vec[axis+1:])

ret\_data.append(reduce\_feat\_vec)

return ret\_data

**Step 1: Calculating Shannon Entropy**

from math import log

import operator

# Calculating Shannon Entropy

def calculate\_entropy(data):

label\_counts = {}

for feature\_data in data:

Laber = feature\_data [-1] The last line is laber

if laber not in label\_counts.keys():

label\_counts[laber] = 0

label\_counts[laber] += 1

count = len(data)

entropy = 0.0

for key in label\_counts:

prob = float(label\_counts[key]) / count

entropy -= prob \* log(prob, 2)

return entropy

**Step 2. A Method of Calculating Information Gain of a Feature**

# Calculating the Information Gain of a Feature

# Index: Which column of data corresponds to the feature to calculate the information gain

# Shannon Entropy of Data

def calculate\_relative\_entropy(data, index, entropy):

Feat\_list= [number [index] for number in data] # Gets all the values under a feature (a column)

uniqual\_vals = set(feat\_list)

new\_entropy = 0

for value in uniqual\_vals:

sub\_data = split\_data(data, index, value)

prob = len(sub\_data) / float(len(data))

New\_entropy += prob \* calculate\_entropy (sub\_data) Summation of Shannon Entropy for each subset

Relative\_entropy = entropy - new\_entropy# Calculates information gain

return relative\_entropy

**Step 3. Selecting features with maximum information gain**

# feature selection for maximum information gain

def choose\_max\_relative\_entropy(data):

num\_feature = len(data[0]) - 1

Base\_entropy = calculate\_entropy (data)# Shannon Entropy

best\_infor\_gain = 0

best\_feature = -1

for i in range(num\_feature):

info\_gain=calculate\_relative\_entropy(data, i, base\_entropy)

# Maximum Information Gain

if (info\_gain > best\_infor\_gain):

best\_infor\_gain = info\_gain

best\_feature = i

return best\_feature

**Step 4. Building Decision Trees**

def create\_decision\_tree(data, labels):

class\_list=[example[-1] for example in data]

# Categories are the same, stop dividing

if class\_list.count(class\_list[-1]) == len(class\_list):

return class\_list[-1]

# The category with the largest number of returns when judging whether all the features have been traversed

if len(data[0]) == 1:

return most\_class(class\_list)

# Selection of Classified Characteristic Attributes Based on Maximum Information Gain

best\_feat = choose\_max\_relative\_entropy(data)

Best\_feat\_lable = labels [best\_feat] label for this feature

Decision\_tree= {best\_feat\_lable: {}} Dictionary for building trees

Del (labels [best\_feat]) Delete the label from the list of labels

feat\_values = [example[best\_feat] for example in data]

unique\_values = set(feat\_values)

for value in unique\_values:

sub\_lables=labels[:]

# Construct a subset of data and recurse it

decision\_tree[best\_feat\_lable][value] = create\_decision\_tree(split\_data(data, best\_feat, value), sub\_lables)

return decision\_tree

Two tool approaches are used in the process of building a decision tree:

# Returns the largest number of categories when traversing all features

def most\_class(classList):

class\_count={}

for vote in classList:

if vote not in class\_count.keys():class\_count[vote]=0

class\_count[vote]+=1

sorted\_class\_count=sorted(class\_count.items,key=operator.itemgetter(1),reversed=True)

return sorted\_class\_count[0][0]

# Tool function inputs three variables (data set, feature, classification value) to return a subset without partition feature

def split\_data(data, axis, value):

ret\_data=[]

for feat\_vec in data:

if feat\_vec[axis]==value :

reduce\_feat\_vec=feat\_vec[:axis]

reduce\_feat\_vec.extend(feat\_vec[axis+1:])

ret\_data.append(reduce\_feat\_vec)

return ret\_data