2021/10/13

Decision Tree

Conceptually, a decision tree is **grown** by first splitting all data points into two groups, such that similar data points are grouped together, and then further repeating this binary splitting process within each group.

As a result, each subsequent leaf node would **have fewer but more homogeneous data points**. The basis of decision trees is that data points following the same path are likely to be similar to each other.

The process of repeatedly splitting data to obtain homogeneous groups is called recursive partitioning. It involves just two steps:

- **Step 1**: Identify the binary question that best splits data points into two groups that are most homogeneous.
- **Step 2**: Repeat Step 1 for each leaf node, until a stopping criterion is reached.

Step 1 involves **best split of two groups** and we will visit the best splitting criteria later. **Step 2** involves recursive stopping criteria. There are various possibilities for stopping criteria; these include:

- Stop when data points at each leaf are all of the same predicted category or value.
- Stop when the leaf contains less than few data points.
- Stop when further branching does not improve homogeneity beyond a minimum threshold.
- Stop when overfitting occurs.

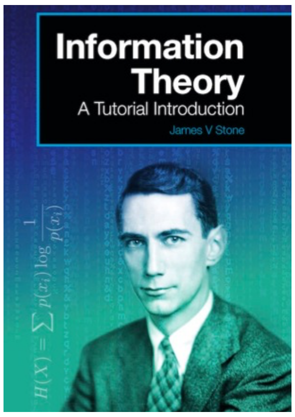
1. Theoretical Minimum: How a Decision Tree Picks Its Split

A key step in decision tree is to identify the binary question that best splits data points into two groups that are most homogeneous.

The obvious question becomes "What is best?"

The **solution** comes from "Information Theory" by Claude Shannon[1]

[1] In 1948, Claude Shannon published a paper called *A Mathematical Theory of Communication*. This paper heralded a transformation in our understanding of information. Before Shannon's paper, information had been viewed as a kind of poorly defined miasmic fluid. But after Shannon's paper, it became apparent that information is a well-defined and, above all, measurable quantity.



The most commonly used solution is either the “Gini” criteria, or the “Entropy” criteria. The next few pages give examples of both equations which are similar, but a little different. However, at the end of the day, it usually makes very little difference which one you use, as they tend to give results that are only a few percent different.

In `Scikit-Learn`, the default is the “Gini” criteria, so we’ll start with that.

<u>Gini Criteria</u>

The equation for the **Gini impurity** is:

$$Gini = 1 - \sum_j p_j^2$$

where p is the probability of having a given data class in your dataset. **The lower the Gini impurity, the better.**

For instance, let’s say that you have a dataset of 10 Apples, 6 Bananas, and 4 Coconuts. The probability for each class is $\frac{10}{20}$ (Apple), $\frac{6}{20}$ (Banana), $\frac{4}{20}$ (Coconut).

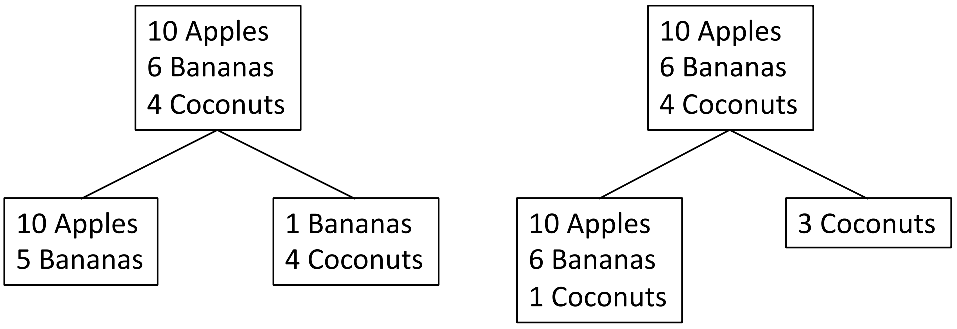
Q: What is the Gini impurity for this case?

A:

$Gini = 1 - \sum_j p_j^2 = (1 - 0.5^2 - 0.3^2 - 0.2^2) = 0.62$

Remark: The best value that we could have is an impurity of 0. That would occur if we had a branch that is 100% one class, since the equation would become 1 – 1.

Now let’s say that the decision tree has two possibilities to split the dataset into two branches:

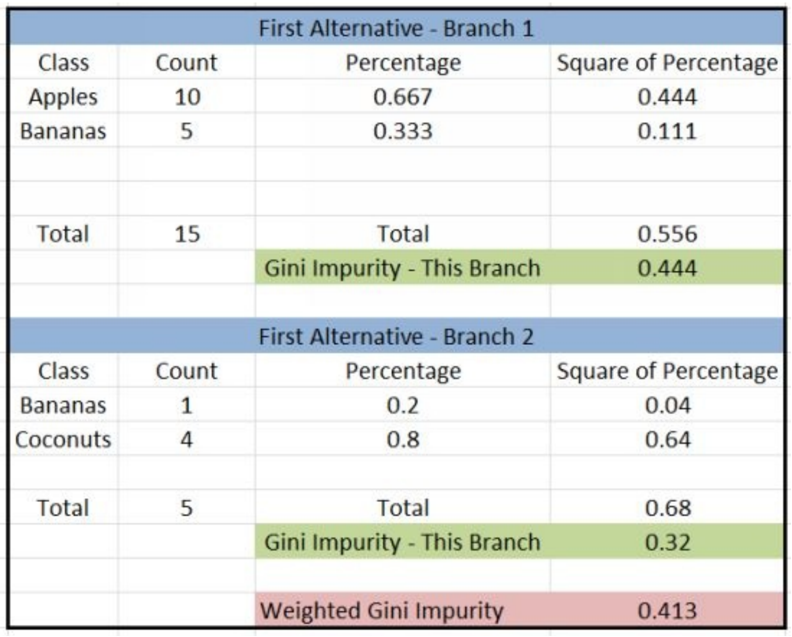


Q: Which one is better?

A: The answer is to calculate the Gini Impurity for both possible splits, and see which one is lower.

First Possible Split:

For the first possible split, we calculate the Gini Impurity of Branch 1 to be .444 and the Gini Impurity of Branch 2 to be .32



First Alternative - Branch 1			
Class	Count	Percentage	Square of Percentage
Apples	10	0.667	0.444
Bananas	5	0.333	0.111
Total	15	Total	0.556
		Gini Impurity - This Branch	0.444
First Alternative - Branch 2			
Class	Count	Percentage	Square of Percentage
Bananas	1	0.2	0.04
Coconuts	4	0.8	0.64
Total	5	Total	0.68
		Gini Impurity - This Branch	0.32
		Weighted Gini Impurity	0.413

There are 15 items in branch 1, and there are 5 items in branch 2. So we can calculate the combined Gini impurity of both branches by taking the weighted average of the two branches = (15*.444+5*.32)/20 which gives a total value of **.413**.

You can do the same thing for the Second Possible Split and obtain a total value of **.447**. So with these two alternatives a decision tree would be generated with the first choice.

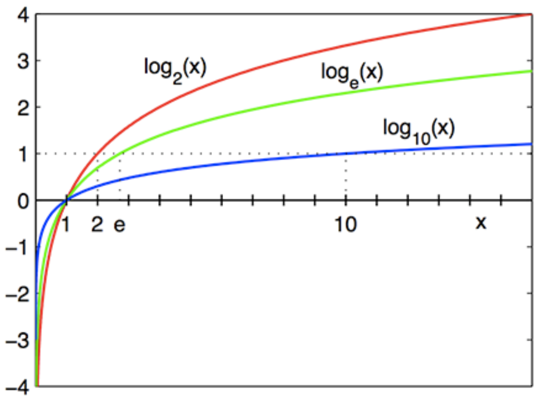
Entropy Criteria

The equation for entropy is different than the Gini equation, but other than that, the process is pretty much the same. The equation for entropy is:

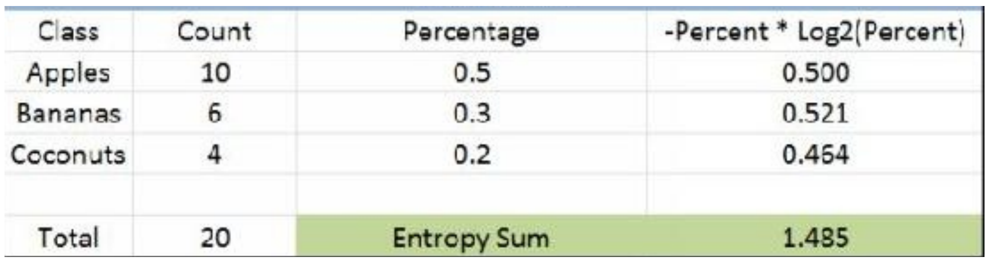
$$Entropy = \sum_j -p_j \times \log_2(p_j)$$

where p is the probability of having a given data class in your dataset. For entropy, just like the Gini criteria, the lower the number the better, with the best being an Entropy of zero.

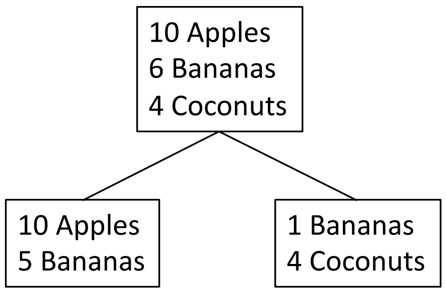
Remark: For this equation, for each probability, we are multiplying that probability by the base 2 logarithm of that probability. Since each of these probabilities are a decimal between 0 and 1, the base 2 logarithm will always be negative (or zero), which when multiplied by the negative sign in the equation will give a positive number for the total entropy summation.

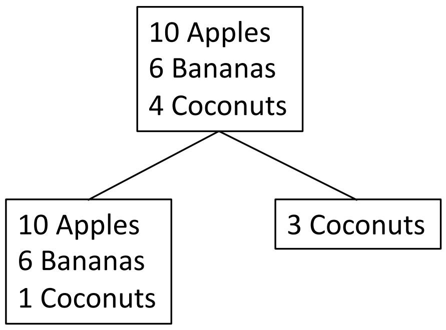


If we go back to the scenario where we have a dataset of 10 Apples, 6 Bananas, and 4 Coconuts. The probability for each class is $\frac{10}{20}$ (Apple), $\frac{6}{20}$ (Banana), $\frac{4}{20}$ (Coconut). We can calculate the total entropy to be 1.485 as show below:

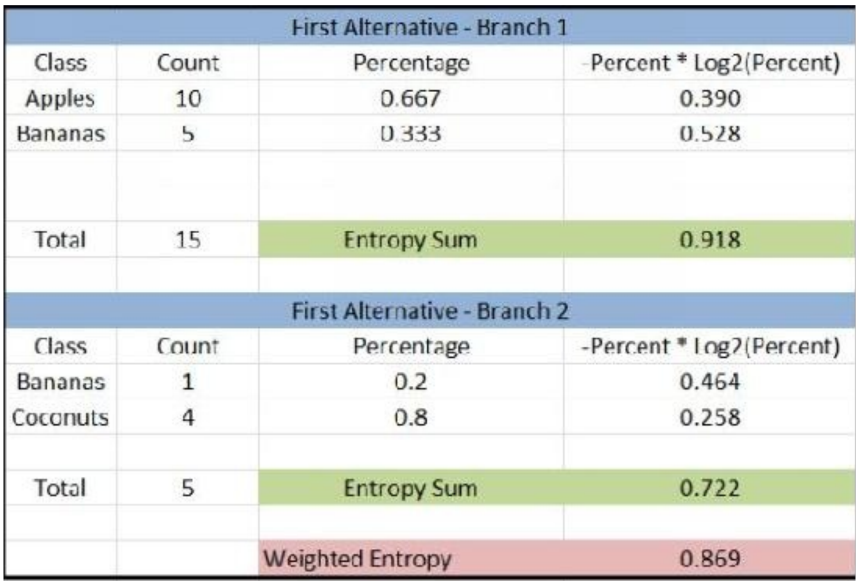


Class	Count	Percentage	-Percent * Log2(Percent)
Apples	10	0.5	0.500
Bananas	6	0.3	0.521
Coconuts	4	0.2	0.464
Total	20	Entropy Sum	1.485





For first possibility of split, we can calculate the weighted entropy to be **.869** (see below) and for second possibility, the weighted entropy is **1.038**.



First Alternative - Branch 1			
Class	Count	Percentage	-Percent * Log2(Percent)
Apples	10	0.667	0.390
Bananas	5	0.333	0.528
Total	15	Entropy Sum	0.918
First Alternative - Branch 2			
Class	Count	Percentage	-Percent * Log2(Percent)
Bananas	1	0.2	0.464
Coconuts	4	0.8	0.258
Total	5	Entropy Sum	0.722
		Weighted Entropy	0.869

So for this example, just like for the Gini criteria, we see that the first possible split has a lower entropy than the second possible split, so the first possibility would be the branching that was generated.

2. Selecting an Attribute Test Condition

At each recursive step in a decision tree, **an attribute (feature) must be selected** to partition the training instances associated with a node into smaller subsets associated with its child nodes. Both Gini and Entropy measures can be used to determine the goodness of an attribute test condition.

To determine the goodness of an attribute test condition, we need to compare the degree of impurity of the parent node (before splitting) with the weighted degree of impurity of the child nodes (after splitting). The larger their difference, the better the test condition. This difference, Δ, also termed as **the gain in purity** of an attribute test condition, can be defined as follows:

$$\Delta = I_{parent} - I_{children}$$

where I_{parent} is the impurity of a node before splitting and $I_{children}$ is the weighted impurity measure after splitting. The higher the gain, the purer are the classes in the child nodes relative to the parent node.

The splitting criterion in the decision tree learning algorithm selects the attribute test condition that shows the maximum gain.

Remark: Note that maximizing the gain at a given node is equivalent to minimizing the weighted impurity measure of its children since I_{parent} is the same for all candidate attribute test conditions. Finally, when entropy is used as the impurity measure, the difference in entropy is commonly known as information gain, Δ_{info}.

To illustrate how this works, consider again the training set shown in the Table 4.1 for the loan borrower classification problem.

Table 4.1 A sample data for the loan borrower classification problem.

ID	Home Owner	Marital Status	Annual Income	Defaulted?
1	Yes	Single	125000	No
2	No	Married	100000	No
3	No	Single	70000	No
4	Yes	Married	120000	No
5	No	Divorced	95000	Yes
6	No	Single	60000	No
7	Yes	Divorced	220000	No
8	No	Single	85000	Yes
9	No	Married	75000	No
10	No	Single	90000	Yes

Consider building a decision tree using only binary splits and the Gini impurity as the measure. Recall the equation for the **Gini impurity** is:

$$Gini = 1 - \sum_j p_j^2$$

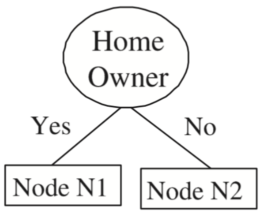
where p is the probability of having a given data class in your dataset.

Q: What is the Gini impurity of the parent node before splitting? (**hint**: in our data class, we have 3 borrowers in the training set who defaulted and 7 others who repaid their loan)

A: Since there are 3 borrowers in the training set who defaulted and 7 others who repaid their loan, the

Gini index of the parent node before splitting is $Gini = 1 - \sum_j p_j^2 = (1 - 0.3^2 - 0.7^2) = 0.42$

Q: If we split the node based on the attribute of `Home Owner`, what is the weighted Gini impurity of the child nodes?



	N1	N2
No	**3**	**4**
Yes	**0**	**3**

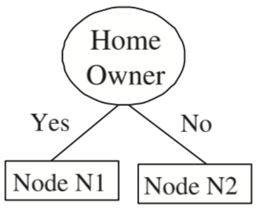
A:

$$Gini = \frac{3}{10} * \left(1 - \left(\frac{3}{3}\right)^2\right) + \frac{7}{10} * \left(1 - \left(\frac{4}{7}\right)^2 - \left(\frac{3}{7}\right)^2\right) = 0.343$$

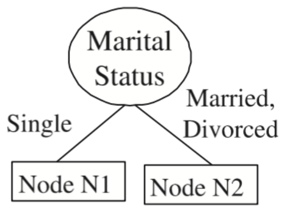
The gain using `Home Owner` as splitting attribute is 0.420 - 0.343 = 0.077. Similarly, we can apply a binary split on the `Marital Status` attribute. However, since `Marital Status` is a nominal attribute with three outcomes, there are three possible ways to group the attribute values into a binary split.

The weighted average Gini index of the children for each candidate binary split is shown below. Based on these results, `Home Owner` and the last binary split using `Marital Status` are clearly the best candidates, since they both produce the lowest weighted average Gini impurity or produce the highest gain after splitting.

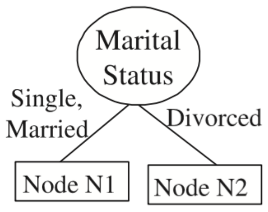
	Parent
No	**7**
Yes	**3**
Gini = 0.420	



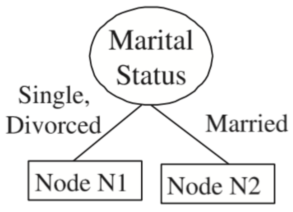
	N1	N2
No	**3**	**4**
Yes	**0**	**3**
Gini= 0.343		



	N1	N2
No	**3**	**4**
Yes	**2**	**1**
Gini= 0.400		



	N1	N2
No	**6**	**1**
Yes	**2**	**1**
Gini= 0.400		



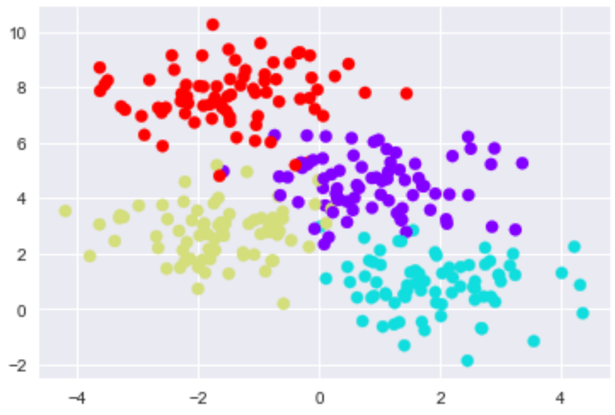
	N1	N2
No	**4**	**3**
Yes	**3**	**0**
Gini= 0.343		

3. `Sckit-Learn` Example

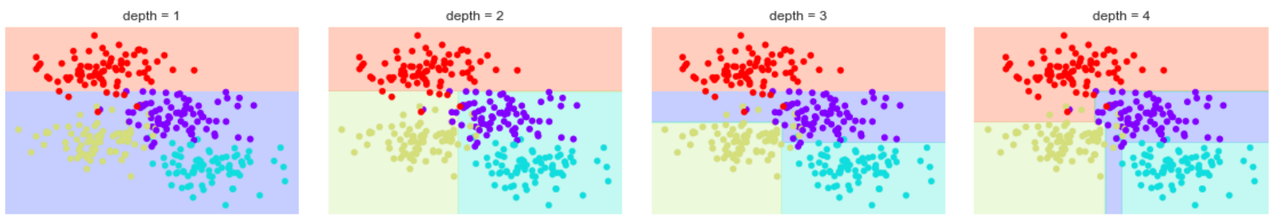
Consider the following two-dimensional data, which has one of four class labels:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=300, centers=4,
                  random_state=0, cluster_std=1.0)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='rainbow');
```



A simple decision tree built on this data will iteratively split the data along one or the other axis according to some quantitative criterion. Let us visualize the first four levels of a decision tree classifier for this data:



Below are the python codes to accomplish the task (including a utility function `visualize_classifier` to help us visualize the output of the classifier:):

```
def visualize_classifier(model, X, y, ax=None, cmap='rainbow'):
    ax = ax or plt.gca()

    # Plot the training points
    ax.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=cmap,
              clim=(y.min(), y.max()), zorder=3)
    ax.axis('tight')
    ax.axis('off')
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # fit the estimator
    model.fit(X, y)
    xx, yy = np.meshgrid(np.linspace(*xlim, num=200),
                         np.linspace(*ylim, num=200))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    # Create a color plot with the results
    n_classes = len(np.unique(y))
    contours = ax.contourf(xx, yy, Z, alpha=0.3,
                           levels=np.arange(n_classes + 1) - 0.5,
                           cmap=cmap, clim=(y.min(), y.max()),
                           zorder=1)
    ax.set(xlim=xlim, ylim=ylim)

from sklearn.tree import DecisionTreeClassifier

fig, ax = plt.subplots(1, 4, figsize=(16, 3))
fig.subplots_adjust(left=0.02, right=0.98, wspace=0.1)

for axi, depth in zip(ax, range(1, 5)):
    model = DecisionTreeClassifier(max_depth=depth)
    visualize_classifier(model, X, y, ax=axi)
    axi.set_title('depth = {0}'.format(depth))
```

Remark: Notice that as the depth increases, we tend to get very strangely shaped classification regions; for example, at a depth of four, there is a tall and skinny purple region between the yellow and blue regions. It's clear that this is less a result of the intrinsic data distribution, and more a result of the particular sampling or noise properties of the data. That is, this decision tree, even at only four levels deep, is clearly **over-fitting** our data.

Q: What is the problem of overfitting?
A: Even if a model fits well over the training data, it can still show poor generalization performance, a phenomenon known as model overfitting.

You can download the above Python codes `Decision_tree-tutorial.ipynb` from the course website.

4. Decision Tree and Over-fitting

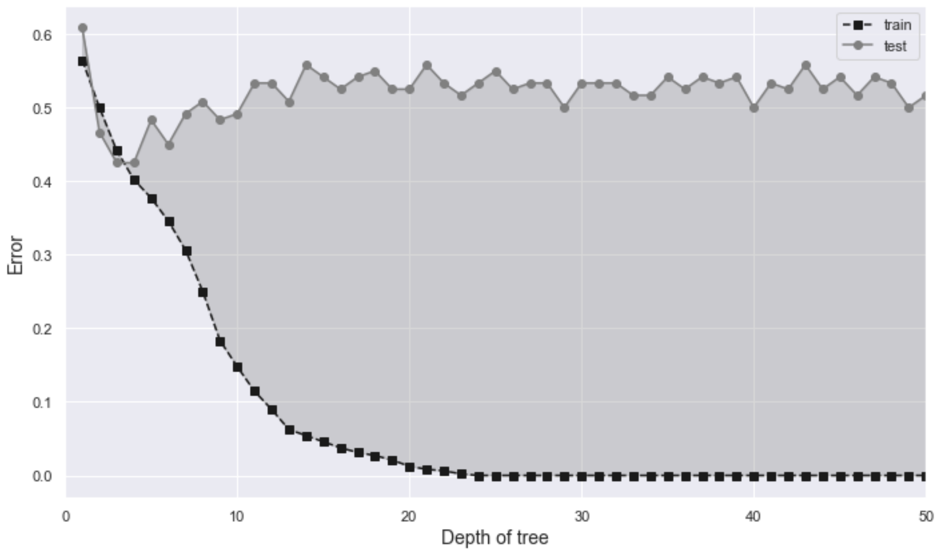
The over-fitting turns out to be a general property of decision trees: it is very easy to go too deep in the tree, and thus to fit details of the particular data rather than the overall properties of the distributions they are drawn from. This will lead to good fit over the training data, but poor generalization performance.

<u>Over-fitting Example</u>: Let us consider a dataset with 600 sample points. Spilt the data into 80% training and 20% testing sizes and report the training and testing scores from decision tree classifier vs. the depth of the tree.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.datasets import make_blobs
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

X, y = make_blobs(n_samples=600, centers=4,
               random_state=0, cluster_std = 3.0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state=42)
otrn = []
otst = []
tree_depth = 51
for depth in range(1, tree_depth):
   tree = DecisionTreeClassifier(max_depth=depth)
   tree.fit(X_train, y_train)
   otrn.append(1-tree.score(X_train, y_train))
   otst.append(1-tree.score(X_test, y_test))

divisors = range(1, tree_depth)
fig,ax=plt.subplots()
fig.set_size_inches((10,6))
_=ax.plot(divisors,otrn,'--s',label='train',color='k')
_=ax.plot(divisors,otst,'-o',label='test',color='gray')
_=ax.fill_between(divisors,otrn,otst,color='gray',alpha=.3)
_=ax.legend(loc=0)
_=ax.set_xlabel('Depth of tree',fontsize=14)
_=ax.set_ylabel('Error',fontsize=14)
_=ax.axis(xmin=0,xmax=50)
fig.tight_layout()
```



As the depth increases, we clearly observe a very good fit over the training data, but poor generalization performance. You can download the above Python codes `DT_overfit.ipynb` from the course website.

5. Decision Tree Classifier: Summary

- A decision tree makes predictions by asking a sequence of binary questions.
- The data sample is split repeatedly to obtain homogeneous groups in a process called recursive partitioning, until a stopping criterion is reached. At each recursive step in a decision tree, **an attribute must be selected** to partition the training instances associated with a node into smaller subsets associated with its child nodes. The "Gini" criteria, or the "Entropy" criteria from Information Theory is the most commonly used solution to determine **the best split**.
- **While easy to use and understand**, decision trees are prone to overfitting, which leads to inconsistent results. To minimize this, we could use an ensemble of randomized decision trees such as random forests. This will be covered in the sequel.