Module V - Machine Learning

siju.swamy@saintgits.org

August 29, 2022

1 Introduction

We are entering the era of big data. For example, there are about 1 trillion web pages; one hour of video is uploaded to YouTube every second, amounting to 10 years of content every day; the genomes of 1000s of people, each of which has a length of 3.8×10^9 base pairs, have been sequenced by various labs; Walmart handles more than 1M transactions per hour and has databases containing more than 2.5 petabytes (2.5×10^{15}) of information (Cukier 2010); and so on.

This deluge of data calls for automated methods of data analysis, which is what machine learning provides. This context prompts the concept of learning from Examples. We describe agents that can improve their behaviour through diligent study of past experiences and predictions about the future. An agent is *learning* if it improves its performance after making observations about the world. Learning can range from the trivial, such as copying a number, to the identification of a new star in the galaxy. When the agent is a computer, we call it machine learning: a computer observes some data, builds a model based on the data, and uses the model as both a hypothesis about the world and a piece of software that can solve problems. Formally the machine learning is defined as follows(Murphy,2006).

Definition: 1 (Machine Learning) *Machine learning is a family of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty.*

1.1 Why Machine Learning?

There are two main reasons. First, the designers cannot anticipate all possible future situations. For example, a robot designed to navigate mazes must learn the layout of each new maze it encounters; a program for predicting stock market prices must learn to adapt when conditions change from boom to bust. Second, sometimes the designers have no idea how to program a solution themselves. Most people are good at recognizing the faces of family members, but they do it subconsciously, so even the best programmers don't know how to program a computer to accomplish that task, except by using machine learning. algorithms.

2 Types of machine learning

Machine learning is usually divided into two main types- Supervised and unsupervised learning.

2.1 Supervised learning

In the predictive or supervised learning approach, the goal is to learn a mapping f from inputs x to outputs y, given a labelled set of input-output pairs $\mathscr{D} = \{(x_i, y_i)\}_{i=1}^N$. Here $\mathscr{D}$ is called the training set, and N is the number of training examples [1]. Where each pair was generated by an unknown function $y = f(x)$. The job is to discover a function h that approximates the true function f. The function h is called a hypothesis about the world. It is drawn from a hypothesis space **H** of possible functions. In technical language, we can say that h is a model of the data, drawn from a model class **H** or we can say a function drawn from a function class. We call the output y_i the ground truth—the true answer we are asking our model to predict.

[1] In supervised learning the agent observes input-output pairs and learns a function that maps from input to output. For example, the inputs could be camera images, each one accompanied by an output saying "bus" or "pedestrian," etc. An output like this is called a label. The agent learns a function that, when given a new image, predicts the appropriate label.

2.1.1 Examples

Some real life examples of classification problems are:

1. Document classification
2. Spam detection in emails
3. Image classification and handwriting recognition
4. Face detection and recognition

In the simplest setting, each training input x_i is a D-dimensional vector of numbers, representing, say, the height and weight of a person. These are called features, attributes or covariates. In general, however, x_i could be a complex structured object, such as an image, a sentence, an email message, a time series, a molecular shape, a graph, etc. Similarly the form of the output or response variable can in principle be anything, but most methods assume that y_i is a categorical or nominal variable from some finite set, $y_i \in \{1, \ldots, C\}$ (such as male or female), or that y_i is a real-valued scalar (such as income level). When y_i is categorical, the problem is known as *classification* or pattern recognition, and when y_i is real-valued, the problem is known as *regression*. Another variant, known as *ordinal regression*, occurs where label space $\mathcal{Y}$ has some natural ordering, such as grades A–F.

2.2 Examples

Here are some examples of real-world regression problems.

1. Predict tomorrow's stock market price given current market conditions and other possible side information.
2. Predict the age of a viewer watching a given video on YouTube.
3. Predict the location in 3d space of a robot arm end effector, given control signals (torques) sent to its various motors.
4. Predict the amount of prostate specific antigen (PSA) in the body as a function of a number of different clinical measurements.
5. Predict the temperature at any location inside a building using weather data, time, door sensors, etc.

2.3 Unsupervised learning

The second main type of machine learning is the descriptive or unsupervised learning approach. Here we are only given inputs, $\mathcal{D} = \{(x_i)\}_{i=1}^N$, and the goal is to find "interesting patterns" in the data. This is sometimes called knowledge discovery. This is a much less well-defined problem, since we are not told what kinds of patterns to look for, and there is no obvious error metric to use (unlike supervised learning, where we can compare our prediction of y for a given x to the observed value). In unsupervised learning the agent learns patterns in the input without any explicit feedback. The most common unsupervised learning task is clustering: detecting potentially useful clusters of input examples. For example, when shown millions of images taken from the Internet, a computer vision system can identify a large cluster of similar images which an English speaker would call "cats."

There is a third type of machine learning, known as reinforcement learning. In reinforcement learning the agent learns from a series of reinforcements: rewards and punishments. For example, at the end of a chess game the agent is told that it has won (a reward) or lost (a punishment). It is up to the agent to decide which of the actions prior to the reinforcement were most responsible for it, and to alter its actions to aim towards more rewards in the future.

3 Identifying the Best Function Approximation-h^* from the Hypothesis Space-H

A good hypothesis should be consistent with the inputs. A consistent hypothesis- h is one such that each x_i in the training set $\mathscr{D}$ has the property, $h(x_i) = y_i$ With continuous-valued outputs we can't expect an exact match to the ground truth; instead we look for a *best-fit* function for which each $h(x_i)$ is close to y_i.

The true measure of a hypothesis is not how it does on the training set, but rather how well it handles inputs it has not yet seen. We can evaluate that with a second sample of (x_i, y_i) pairs called a test set. We say that h generalizes well if it accurately predicts the outputs of the test set. A consistent hypothesis that generalizes the input-output mapping well.

Discovery of the learning algorithm depends on the hypothesis space **H** it considers and on the training set it is given. To illustrate the behaviour of a hypothesis on various sub-samples of the same dataset, following univariate data distribution in the 2D plane is considered (Refer Figure 1).

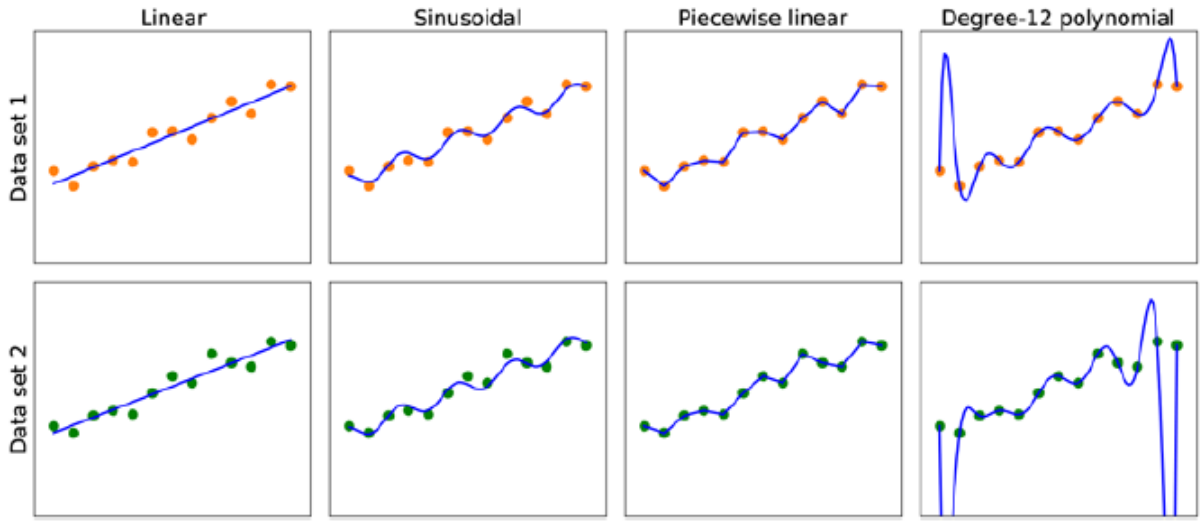


Figure 1: Performance of hypothesis on sub samples

Each of the four plots in the top row have the same training set of 13 data points in the (x, y) plane. The four plots in the bottom row have a second set of 13 data points; both sets are representative of the same unknown function $f(x)$. Each column shows the best-fit hypothesis from a different hypothesis space. The selected hypothesis in each case is listed bellow:

COLUMN 1: Straight lines; functions of the form $h(x) = \theta_1 x + \theta_0$. There is no line that would be a consistent hypothesis for the data points.

COLUMN 2: Sinusoidal functions of the form $h(x) = \theta_1 x + \sin(\theta_0 x)$ This choice is not quite consistent, but fits both data sets very well.

COLUMN 3: Piecewise-linear functions where each line segment connects the dots from one data point to the next. These functions are always consistent.

COLUMN 4: Degree-12 polynomials,$h(x) = \sum_{i=0}^{12} \theta_i x^i$ These are consistent: we can always get a degree-12 polynomial to perfectly fit 13 distinct points. But just because the hypothesis is consistent does not mean it is a good guess.

3.1 Bias and variance of a hypothesis

One way to analyze hypothesis spaces is by the bias they impose (regardless of the training data set) and the variance they produce (from one training set to another).

By *bias* we mean the tendency of a predictive hypothesis to deviate from the expected value when averaged over different training sets. Bias often results from restrictions imposed by the hypothesis space. For example, the hypothesis space of linear functions induces a strong bias: it only allows functions consisting of straight lines. If there are any patterns in the data other than the overall slope of a line, a linear function will not be able to represent those patterns. We say that a hypothesis is underfitting when it fails to find a pattern in the data. On the other hand, the piecewise linear function has low bias; the shape of the function is driven by the data.

By *variance* we mean the amount of change in the hypothesis due to fluctuation in the training data. The two rows of Figure 1 represent data sets that were each sampled from the same function $f(x)$. The data sets turned out to be slightly different. For the first three columns, the small difference in the data set translates into a small difference in the hypothesis. We call that low variance. But the degree-12 polynomials in the fourth column have high variance: look how different the two functions are at both ends of the -axis. Clearly, at least one of these polynomials must be a poor approximation to the true $f(x)$. We say a function is overfitting the data when it pays too much attention to the particular data set it is trained on, causing it to perform poorly on unseen data. Often there is a bias–variance tradeoff: a choice between more complex, low-bias hypotheses that fit the training data well and simpler, low-variance hypotheses that may generalize better.

Supervised learning can be done by choosing the hypothesis h^* that is most probable given the data:

$$h^* = \underset{h \in H}{\operatorname{argmax}} P(h|Data) \tag{1}$$

In the language of machine learning, we will train the models with proper validation splits. It will help us to identify the under-fitting/ over-fitting. Finally based on the performance measures, we will fix the best fit model. This process is called the model selection.

4 Examples of Supervised ML Algorithms

To understand the working of Machine Learning models, two simple supervised models will be discussed in this section.

4.1 Linear Regression Models

The simplest form of supervised learning is the regression models. Since regression is the modified version of the curve fitting in Mathematics, it is easy to comprehend the underlying math.

Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used.

4.1.1 Linear Regression Job

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(response). Hence, the name is Linear Regression. For example X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our SLR model.

4.1.2 Hypothesis of SLRM

The proposed model is

$$h_\theta(x) = \theta_0 + \theta_1 x$$

, where θ_0 is the intercept of the line and θ_1 is the slope. In case of multiple linear regression the vector form of LR is

$$h_\theta = \theta^T X + \theta_0$$

where $\theta = (\theta_1, \theta_2 \ldots, \theta_n)^T$
The hypothesis are:

$$H_0 : \theta_1 = 0$$

$$H_1 : \theta_1 \neq 0$$

While training the model we are given :

1. x: input training data (univariate – one input variable(parameter))
2. y: labels to data (supervised learning)

When training the model , it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ_0 and θ_1 values. θ_0: intercept θ_1: coefficient of x

4.2 Moving in SLR model building

Once we find the best θ_0 and θ_1 values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x. There are many ways create a SLR model. Let us discuss the classical methods and finally the machine learning approach.

4.2.1 Least square error method

In this method, we simply calculate the parameters θ_0 and θ_1 using the following algebraic expressions:

$$\theta_1 = \frac{\sum_{i=1}^{m} (x_i - \bar{x})\left(y_i - \bar{y}\right)}{\sum_{i=1}^{m} (x_i - \bar{x})^2}$$

$$\theta_0 = \bar{y} - \theta_1 \bar{x}$$

This is purely a tabular method. Using a spreadsheet program, we can calculate θ_0 and θ_1.

4.2.2 `Python` implementation of Least square error method

```
import numpy as np
x=np.array([1,2,3,4,5,6,7,8])
y=4*x
nr=np.sum((np.multiply(y-np.mean(y),x-np.mean(x))))
dnr=np.sum((x-np.mean(x))**2)
# print(nr)
# print(dnr)
theta_1=nr/dnr
theta_0=np.mean(y)-theta_1*np.mean(x)
print("slope :%d"%theta_1)
print("Intercept :%d"%theta_0)
```

4.2.3 Direct Statistical method

The `statmodel` python library provides a direct method to calculate regression coefficients with sufficient model fit measures. The following code chunk demonstrate the implementation.

```
import statsmodels.api as sm
import pandas
# Note the difference in argument order
model = sm.OLS(y, x).fit()
predictions = model.predict(x) # make the predictions by the model

# Print out the statistics
model.summary()
```

Output of the above code is shown in Figure

OLS Regression Results

Dep. Variable:	y	**R-squared:**	1.000
Model:	OLS	**Adj. R-squared:**	1.000
Method:	Least Squares	**F-statistic:**	3.954e+32
Date:	Wed, 24 Aug 2022	**Prob (F-statistic):**	2.15e-112
Time:	08:13:34	**Log-Likelihood:**	257.05
No. Observations:	8	**AIC:**	-512.1
Df Residuals:	7	**BIC:**	-512.0
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
x1	4.0000	2.01e-16	1.99e+16	0.000	4.000	4.000

Omnibus:	2.672	**Durbin-Watson:**	0.072
Prob(Omnibus):	0.263	**Jarque-Bera (JB):**	0.939
Skew:	-0.320	**Prob(JB):**	0.625
Kurtosis:	1.448	**Cond. No.**	1.00

For the statistical analysis of linear association of one variable on the other, we will use the above method. Still it is not a ML model. It just provides coefficients of the regression model and its fit measures. If we want to use this coefficients for prediction, we have to evaluate the responses mathematically. Ordinary Least Square method looks simple and computation is easy. But, this OLS method will only work for a univariate dataset consist of single independent variable and single dependent variable. Multi-variate dataset contains a single dependent variable and multiple independent variables sets, demands the use a machine learning algorithm called “Gradient Descent”.

4.3 Optimizing Regression line- Gradient descent method

We start with the hypothesis $y = h_\theta(x) = \theta_0 + \theta_1 x$ be a linear approximation for the underlying relationship between x and y. Our objective is to find the values of θ_0 and θ_1 which fit to a best approximation. While learning the linear regression , two functions are introduced:

1. the cost function
2. gradient descent updation

4.3.1 What is a Cost Function?

In the case of Linear Regression, the objective is to find a line of best fit for some given inputs, or X values, and any number of Y values, or outputs. A cost function is defined as:

> ...a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. –from Wikipedia

In this situation, the cost is the difference between the ground truth and estimated values, or the hypothesis and the real values–the actual data we are trying to fit a line.

The cost function in the case of SLR is the MSE in prediction. Formally it can be defined as:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^i) - \hat{y}^i \right)^2 \tag{2}$$

Using Gradient descent algorithm , we will figure out a minimal cost function by applying various parameters for θ_0 and θ_1 and see the slope intercept until it reaches convergence.

In a real world example, it is similar to find out a best direction to take a step downhill.

We take a step towards the direction to get down. From the each step, you look out the direction again to get down faster and downhill quickly. The similar approach is using in this algorithm to minimise cost function.

4.3.2 Calculating Gradient Descent for Linear Regression

The gradient descent for the SLR cost function can be found using Classical calculus method. Since the cost function contains the parameters θ_0 and θ_1, the partial derivative corresponding to these parameters together defines the gradient. Mathematically

$$\frac{\partial}{\partial \theta_0} \left(J(\theta_0, \theta_1) \right) = \frac{\partial}{\partial \theta_0} \left(\frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^i) - \hat{y}^i \right)^2 \right) \tag{3}$$

$$\frac{\partial}{\partial \theta_1} \left(J(\theta_0, \theta_1) \right) = \frac{\partial}{\partial \theta_1} \left(\frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^i) - \hat{y}^i \right)^2 \right) \tag{4}$$

To apply rate of change values for θ_0 and θ_1, the below are the equations for θ_0 and θ_1 to apply it on each epoch.

$$\theta_0' = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^i) - \hat{y}^i \right) \tag{5}$$

$$\theta_1' = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^i) - \hat{y}^i \right) x^i \tag{6}$$

By achieving the best-fit regression line, the model aims to predict y value such that the error difference between predicted value and true value is minimum. So, it is very important to update the θ_0 and θ_1 values, to reach the best value that minimize the error between predicted y value ($h_\theta(x)$) and true y value ($\hat{y}$).

To find the best minimum cost, repeat steps to apply various values for θ_0 and θ_1. In other words, repeat steps until convergence evaluate the following:

$$\theta_0^{(i+1)} = \theta_0^i - \eta \theta_0'^{(i)}$$
$$\theta_1^{(i+1)} = \theta_1^i - \eta \theta_1'^{(i)}$$

where η is the learning rate.

4.4 Multiple Linear regression

In the case of multiple linear regression, the hypothesis is

$$y = h_\theta(X) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n + = X^T\theta$$

we have an input vector X and the ground truth is $\hat{y}$. The cost function in the case of MLR is the MSE in prediction. Formally it can be defined as:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(X^i) - \hat{y}^i\right)^2$$

. Now the minimum direction is found by calculating gradient. Here we will have more partial derivatives like:

$$\theta_0' = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(X^i) - \hat{y}^i\right)$$
$$\theta_1' = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(X^i) - \hat{y}^i\right)x_1^i$$
$$\theta_2' = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(X^i) - \hat{y}^i\right)x_2^i$$
$$\ldots\ldots$$

Now the updates in θ will be summarized as:

$$\theta_0^{(i+1)} = \theta_0^i - \eta\theta_0'^{(i)}$$
$$\theta_1^{(i+1)} = \theta_1^i - \eta\theta_1'^{(i)}$$
$$\theta_2^{(i+1)} = \theta_2^i - \eta\theta_2'^{(i)}$$
$$\ldots\ldots$$

4.5 Python code for simple linear regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt# ensure to use matplot library for
    plotting
np.random.seed(42)# ensure reproducible research
X=2*np.random.rand(100,1)# a column of 100 random numbers
y=4+3*X+np.random.rand(100,1)
X_b=np.concatenate([np.ones((100,1)), X], axis=1)# reformulate X
          eta=0.1 # define learning rate
n_iter=1000 # number of epochs
m=100 # number of samples
theta=np.random.rand(2,1) # create an initial setup for theta
for ind in range(n_iter):
    grad=1/m*X_b.T.dot(X_b.dot(theta)-y)
    theta=theta-eta*grad
print(theta)
y_p=X_b.dot(theta)
#print(y_p.T) # predicted values
#print(y) # actual values
```

4.6 Python code for multiple linear regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt# ensure to use matplot library for
    plotting
np.random.seed(42)# ensure reproducible research
X=2*np.random.rand(100,2)# a column of 100 random numbers
t=np.array([2,1])
y=4+np.dot(X,t.T).reshape(100,1)+np.random.rand(100,1)
X_b=np.concatenate([np.ones((100,1)), X], axis=1)# reformulate X
eta=0.1 # define learning rate
n_iter=1000 # number of epochs
m=100 # number of samples
theta=np.random.rand(3,1) # create an initial setup for theta
for ind in range(n_iter):
    grad=1/m*X_b.T.dot(X_b.dot(theta)-y)
    theta=theta-eta*grad
print(theta)
y_p=X_b.dot(theta)
#print(y_p.T) # predicted values
#print(y) # actual values
[[4.55222768]
[2.00029634]
[0.96414812]]
```

4.7 Linear regression using *Scikit-learn* machine learning library in Python

We know that linear regression is a popular technique and we might as well seen the mathematical equation of linear regression. There are several ways in which we can do that, we can do linear regression using numpy, scipy, stats model and sckit learn. But in this post I am going to use scikit learn to perform linear regression.

Scikit-learn is a powerful Python module for machine learning. It contains function for regression, classification, clustering, model selection and dimensionality reduction. We will use the `sklearn.linear_model` module which contains "methods intended for regression in which the target value is expected to be a linear combination of the input variables".

Important points: Important functions to keep in mind while fitting a linear regression model are:

- `lm.fit()` -> fits a linear model
- `lm.predict()` -> Predict Y using the linear model with estimated coefficients
- `lm.score()` -> Returns the coefficient of determination (R^2). A measure of how well observed outcomes are replicated by the model, as the proportion of total variation of outcomes explained by the model.

4.7.1 A direct regression problem with sklearn library- Piza price model

Consider a dataset showing the diameter of the piza and the corresponding price:

Diameter:	6	8	10	14	18
Price:	7	9	13	17.5	18

Fit a linear regression model and explain the fit of the model. **: Python code:**

```
#reading data
import numpy as np
X=[6,8,10,14,18]# reading as a list
Y=[7,9,13,17.5,18]
X=np.array(X).reshape(5,1) # converting list to a numpy array
Y=np.array(Y).reshape(5,1)
```

```
#Visualizing as a scatter plot using Matplotlib
import matplotlib.pyplot as plt# ensure to use matplot library for
    plotting
plt.style.use('seaborn-whitegrid')
plt.xlabel(" Size of piza")
plt.ylabel(" Prize of piza")
plt.scatter(X,Y)
plt.plot(X,Y,'r')
plt.show()
```

4.7.2 Finding linear regression model parameters statistically

$var(x) = \frac{\sum_{i=1}^{n}(x-\bar{x})^2}{n-1}$

$\theta_1 = \frac{cov(x,y)}{var(x)}$;

$\theta_0 = \bar{y} - \theta_1 * \bar{x}$

$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$

$SS_{res} = \sum_{i=1}^{n}(y_i - f(x_i))^2$

$SS_{tot} = \sum_{i=1}^{n}(y_i - \bar{y})^2$

```
# Finding linear regression model parameters statistically
xL=[i[0] for i in X]
yL=[i[0] for i in Y]
var_x=np.var(xL,ddof=1)# d degrees of freedom
cov_xy=np.cov(xL,yL,ddof=1)[0][1]
print(var_x)
print(cov_xy)
xbar=np.mean(xL)
ybar=np.mean(yL)
theta_1=cov_xy/var_x
theta_0=ybar-(theta_1*xbar)
print(theta_0)
y_p=theta_0+theta_1*X
```

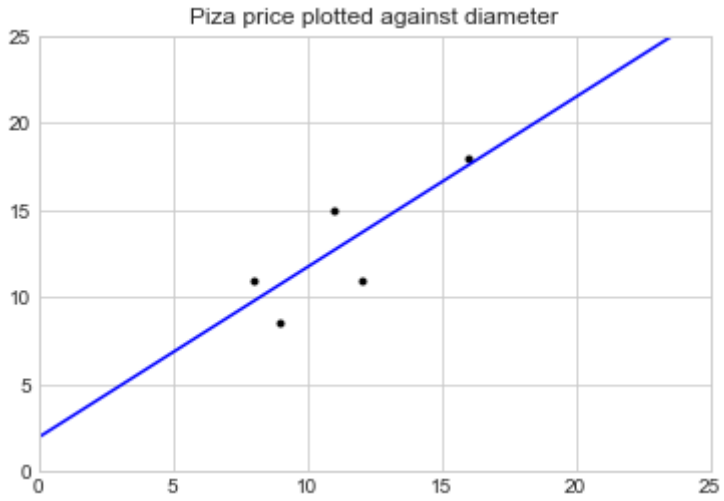
```
#Creating the model and finding parameters
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(X,Y)
print('estimated Intercept:',model.intercept_)
print('number of coefficients',len(model.coef_))
print(' coefficients',model.coef_)
#output:
#('estimated Intercept:', array([1.96551724]))
#('number of coefficients', 1)
#(' coefficients', array([[0.9762931]]))
```

```
#Predict using scikit learn library
x_test=np.array([8,9,11,16,12]).reshape(5,1)
y_test=np.array([11,8.5,15,18,11]).reshape(5,1)
yp_test=model.predict(x_test)
print(yp_test)
print(y_test)
```

```
## Finding skill of the model
print(model.score(X,Y))
##0.9100015964240102
```

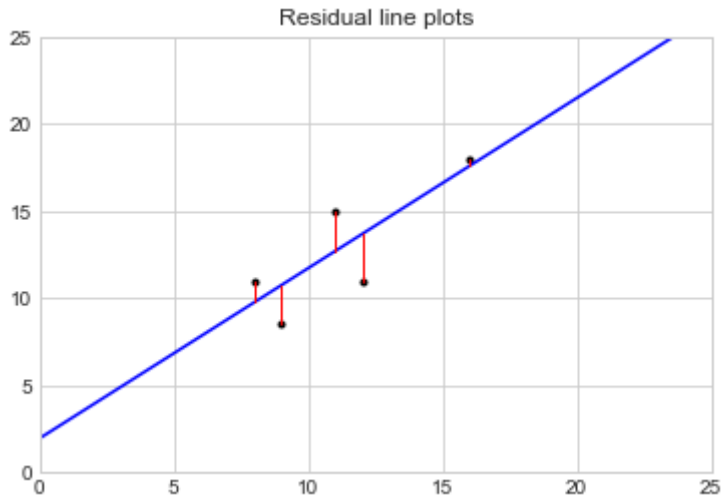
```
##plotting the fitted line using test data
plt.title("Piza price plotted against diameter")
x_fit=np.linspace(0,50)
y_fit=theta_0+theta_1*x_fit
plt.axis([0, 25, 0, 25])
plt.plot(x_test,y_test,'k.')
plt.plot(x_fit,y_fit,color="blue")
plt.show()
```

Output of the above code chunk produce the following plot




```
#Visualizing error in prediction
plt.title("Residual line plots")
plt.axis([0, 25, 0, 25])
plt.plot(x_test,y_test,'k.')
plt.plot(x_fit,y_fit,color="blue")
for i in range(0,len(x_test)):
plt.plot([x_test[i], x_test[i]], [yp_test[i], y_test[i]], color='
   red', linewidth=1)
plt.show()
```

Output of the above code chunk produce the following plot



5 Decision Trees

A decision tree is a representation of a function that maps a vector of attribute values to a single output value–a "decision." A decision tree reaches its decision by performing a sequence of tests, starting at the root and following the appropriate branch until a leaf is reached. Each internal node in the tree corresponds to a test of the value of one of the input attributes, the branches from the node are labeled with the possible values of the attribute,and the leaf nodes specify what value is to be returned by the function.

In general, the input and output values can be discrete or continuous, but for now we will consider only inputs consisting of discrete values and outputs that are either true (a positive example) or false (a negative example). We call this Boolean classification. We will use j to index the examples (x_j is the input vector for the th example and y_j is the output), and $x_{j,i}$ for the i^{th} attribute of the j^{th} example.

The whole Decision tree expression is in disjunctive normal form, which means that any function in propositional logic can be expressed as a decision tree. For many problems, the decision tree format yields a nice, concise, understandable result. For example, the majority function, which returns true if and only if more than half of the inputs are true, requires an exponentially large decision tree, as does the parity function, which returns true if and only if an even number of input attributes are true.

Developing a decision tree with arbitrary root node is need not be the fitted one. So a systematic method is developed for decision tree problems. This algorithm is called the *Learning Decision Tree* algorithm. The pseudocode for this algorithm is shown below.

function LEARN-DECISION-TREE(*examples*, *attributes*, *parent_examples*) **returns** a tree

if *examples* is empty **then return** PLURALITY-VALUE(*parent_examples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
else
  $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, *examples*)
  tree $\leftarrow$ a new decision tree with root test A
  for each value v of A **do**
    $exs \leftarrow \{e : e \in examples \text{ and } e.A = v\}$
    subtree $\leftarrow$ LEARN-DECISION-TREE(exs, $attributes - A$, *examples*)
    add a branch to *tree* with label $(A = v)$ and subtree *subtree*
  return *tree*

The decision tree learning algorithm chooses the attribute with the highest IMPORTANCE. We will now show how to measure importance, using the notion of information gain, which is defined in terms of entropy, which is the fundamental quantity in information theory. Entropy is a measure of the uncertainty of

a random variable; the more information, the less entropy. In general, the entropy of a random variable V with values v_k having $P(v_k)$ probability is defined as:

$$H(V) = -\sum_{v_k \in V} P(v_k) \log_2(P(v_k)) \tag{7}$$

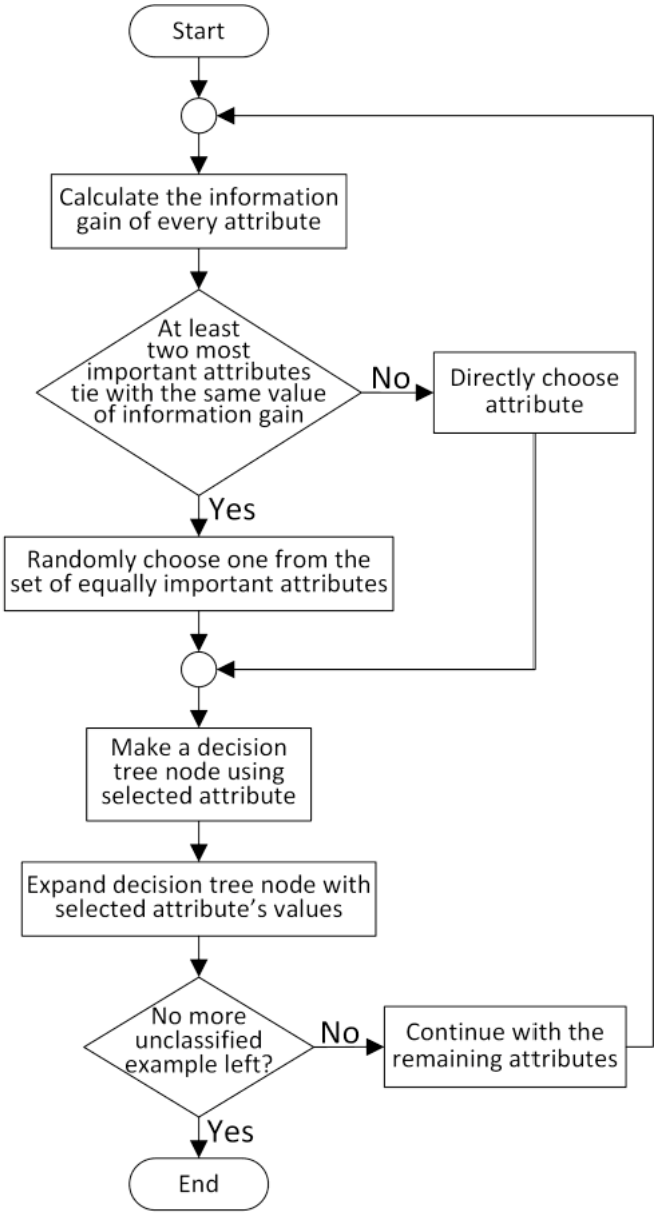
Now we will calculate the Information Gain that determine the importance of an attribute A in decision making.

$$\begin{aligned} Gain(S, A) &= H(S) - H(S/A) \\ &= H(S) - \sum_{v \in V_A} \frac{|S_v|}{|S|} H(S_v) \end{aligned}$$

In fact $Gain()$ is just what we need to implement the IMPORTANCE function in Learning Decision Tree algorithm.

5.1 Flow-chart for Learning Decision Tree Algorithm

Information Gain is the key measure to identify dominant nodes in each level and branching of the tree. A flowchart to implement the LDT algorithm is shown below:



Flowchart of the traditional ID3 algorithm

Example:

From the 9 days of market behaviour a table is created. Past trend, Open interest and Trading volume are the factors affecting the market. Create a decision tree to represent the relationship between the features and market behaviour.

Past Trend	Open Interest	Trading volume	Return
Positive	Low	High	Up
Negative	High	Low	Down
Positive	Low	High	Up
Positive	High	High	Up
Negative	Low	High	Down
Positive	Low	Low	Down
Negative	High	High	Down
Positive	Low	Low	Down
Positive	High	High	Up

Solution: In this case, there are 9 samples and the the response is different(two values). So we have to take the decision based on the features. As the first step we have to identify the most dominant feature that reduce the entropy in decision. The root node will be identified using the information gain measure. Let us denote the 'up' as a positive behaviour and 'Down' as the negative behaviour. Let S denote the sample space of counts of positive and negative responses.

Here $S = [4+, 5-]$

$$\begin{aligned} H(S) &= -\sum P(v_k)\log_2(P(v_k)) \\ &= -\left(\frac{4}{9}\log_2(\frac{4}{9}) + \frac{5}{9}\log_2(\frac{5}{9})\right) \\ &= 0.9910 \end{aligned}$$

Next step is identifying the root node. For this we have to calculate information gain of all attributes in the problem.

LEVEL-0: Root node selection

Attribute: Trading Volume

Values:[High,Low]

$$\begin{aligned} S_{High} &= [4+, 2-] \\ H(S_{High}) &= -\sum P(v_k)\log_2(P(v_k)) \\ &= -\left(\frac{4}{6}\log_2(\frac{4}{6}) + \frac{2}{6}\log_2(\frac{2}{6})\right) \\ &= 0.91829 \\ S_{Low} &= [3-, 0+] \text{(Values are favourable only to up.So Entropy will be 0)} \\ H(S_{Low}) &= -\sum P(v_k)\log_2(P(v_k)) \\ &= -\left(\frac{3}{3}\log_2(\frac{3}{3}) + \frac{0}{3}\log_2(\frac{0}{3})\right) \\ &= 0 \\ Gain(S, TV) &= H(S) - \sum_{v\in\{High,Low\}} \frac{|S_v|}{|S|} H(S_v) \\ &= 0.99107 - (6/9 \times 0.91829 + 3/9 \times 0) \\ &= 0.3787 \end{aligned}$$

Attribute: Opening Interest

Values:[High,Low]

$$
\begin{aligned}
S_{High} &= [2+, 2-] \\
H(S_{High}) &= -\sum P(v_k)\log_2(P(v_k)) \\
&= -\left(\frac{2}{4}\log_2(\frac{2}{4}) + \frac{2}{4}\log_2(\frac{2}{4})\right) \\
&= 1.0 \\
S_{Low} &= [2-, 2+] \\
H(S_{Low}) &= -\sum P(v_k)\log_2(P(v_k)) \\
&= -\left(\frac{2}{4}\log_2(\frac{2}{4}) + \frac{2}{4}\log_2(\frac{2}{4})\right) \\
&= 1.0 \\
Gain(S, OI) &= H(S) - \sum_{v \in \{High, Low\}} \frac{|S_v|}{|S|} H(S_v) \\
&= 0.99107 - (4/9 \times 1 + 4/9 \times 1) \\
&= 0.1021
\end{aligned}
$$

Attribute: Past trend
Values:[Positive,Negative]

$$
\begin{aligned}
S_{Positive} &= [4+, 2-] \\
H(S_{Positive}) &= -\sum P(v_k)\log_2(P(v_k)) \\
&= -\left(\frac{4}{6}\log_2(\frac{4}{6}) + \frac{2}{6}\log_2(\frac{2}{6})\right) \\
&= 0.91829 \\
S_{Negative} &= [3-, 0+] \\
H(S_{Negative}) &= -\sum P(v_k)\log_2(P(v_k)) \\
&= -\left(\frac{3}{3}\log_2(\frac{3}{3}) + \frac{0}{3}\log_2(\frac{0}{3})\right) \\
&= 0.0 \\
Gain(S, PT) &= H(S) - \sum_{v \in \{+ve, -ve\}} \frac{|S_v|}{|S|} H(S_v) \\
&= 0.99107 - (6/9 \times 0.91829 + 3/9 \times 0) \\
&= 0.3787
\end{aligned}
$$

The information gain of all attributes are listed in the following table:

Attribute	Gain(S,A)
Trading Volume	0.3787**
Opening Interest	0.1021
Past Trend	0.3787**

From the gain table, it is clear that highest gain is 0.3787 hold for two attributes-Trading Volume and Past Trend. This tie will brake arbitrarily. Let's choose Past Trend as the root node. So the branching begins at the node Past Trend. The tree at level-0 is:

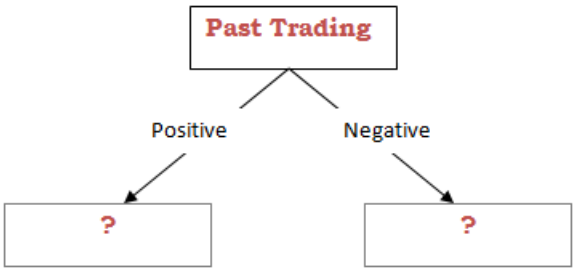


Figure 2: Splitting at the Past Trading attribute.

Level-1

First we identify the attribute in the left branch in Level-1. For this write the updated dataset with 'Past Trading=Positive' after removing the Past Trend column as follows: Here there are only 6 samples in two

Opening Interest	Trading Volume	Return
Low	High	Up
Low	High	Up
High	High	Up
Low	Low	Down
Low	Low	Down
High	High	Up

return values-'Up' and 'Down'. So the revised sample space is $S = [4+, 2-]$.

$$\begin{aligned} H(S) &= -\sum P(v_k)\log_2(P(v_k)) \\ &= -\left(\frac{4}{6}\log_2(\frac{4}{6}) + \frac{2}{6}\log_2(\frac{2}{6})\right) \\ &= 0.91829 \end{aligned}$$

Next step is identifying the decision node. For this we have to calculate information gain of all attributes in the revised dataset.

LEVEL-1: decision node selection

Attribute: Trading Volume

Values:[High,Low]

$$
\begin{aligned}
S_{High} &= [4+, 0-] \\
H(S_{High}) &= -\sum P(v_k)\log_2(P(v_k)) \\
&= -\left(\frac{4}{6}\log_2(\frac{4}{6}) + \frac{0}{6}\log_2(\frac{0}{6})\right) \\
&= 0.0 \\
S_{Low} &= [2-, 0+] \\
H(S_{Low}) &= -\sum P(v_k)\log_2(P(v_k)) \\
&= -\left(\frac{2}{2}\log_2(\frac{2}{2}) + \frac{0}{2}\log_2(\frac{0}{2})\right) \\
&= 0.0 \\
Gain(S, TV) &= H(S) - \sum_{v \in \{High, Low\}} \frac{|S_v|}{|S|} H(S_v) \\
&= 0.91829 - (4/6 \times 0 + 2/6 \times 0) \\
&= 0.91829
\end{aligned}
$$

Attribute: Opening Interest
Values:[High,Low]

$$
\begin{aligned}
S_{High} &= [2+, 0-] \\
H(S_{High}) &= -\sum P(v_k)\log_2(P(v_k)) \\
&= -\left(\frac{2}{2}\log_2(\frac{2}{2}) + \frac{0}{2}\log_2(\frac{0}{2})\right) \\
&= 0.0 \\
S_{Low} &= [2-, 2+] \\
H(S_{Low}) &= -\sum P(v_k)\log_2(P(v_k)) \\
&= -\left(\frac{2}{4}\log_2(\frac{2}{4}) + \frac{2}{4}\log_2(\frac{2}{4})\right) \\
&= 1.0 \\
Gain(S, OI) &= H(S) - \sum_{v \in \{High, Low\}} \frac{|S_v|}{|S|} H(S_v) \\
&= 0.91829 - (2/6 \times 0.0 + 4/6 \times 1.0) \\
&= 0.2516
\end{aligned}
$$

Now the information gain table for Level-1 is: The gain is higher for the attribute- Trading Volume. So it will

Attribute	Gain(S,A)
Trading Volume	0.91829**
Opening Interest	0.2516

be the next decision node to the right of root node. Also note that the 'high' values of the attribute Trading volume give 'up' return and 'low' values give 'down'. There is no further uncertainty. So we can conclude that the Trading Volume is a leaf node. So the left branch of Level-1 is updated as follows:

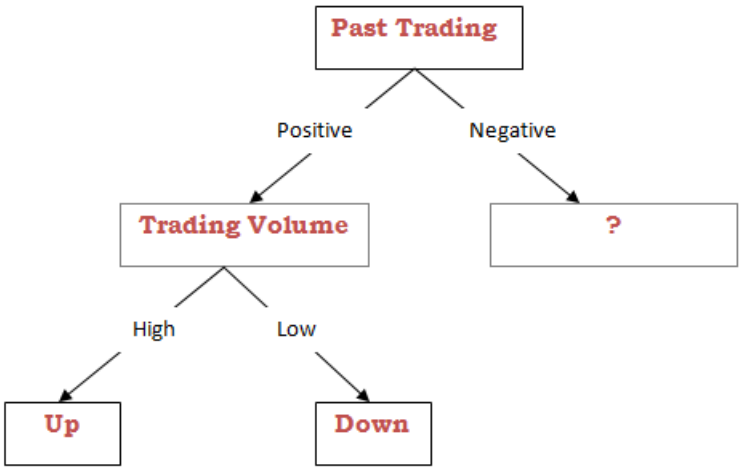


Figure 3: Level-1 update in the left branch

Now consider the right branch of level-1. Here the revised dataset is as shown in the following table. Here note that there is only one value to the return-'Down' irrespective to the values of the attributes. So

Opening Interest	Trading Volume	Return
High	Low	Down
Low	High	Down
High	High	Down

there is no uncertainty in decision. Hence no further branching is needed in the right of the root node. Thus we can assign 'Down' to the right of 'Past Trading' node. The updated tree in level-1 is shown bellow:

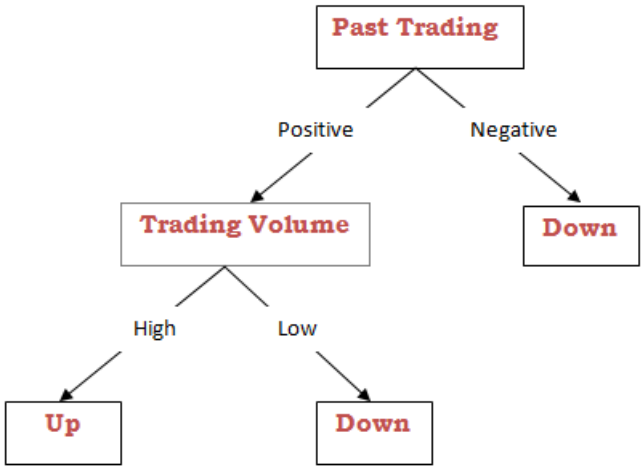


Figure 4: Complete decision tree

6 Computational Approach to Solve a Decision Tree Problem

6.1 LDT algorithm in Python

6.1.1 Loading necessary Libraries

```
import numpy as np
import pandas as pd
eps = np.finfo(float).eps
from numpy import log2 as log
```

6.1.2 Create the dataset for the DT problem

```
#creating data as dictionaries
outlook = 'overcast,overcast,overcast,overcast,rainy,rainy,rainy,
   rainy,rainy,sunny,sunny,sunny,sunny,sunny'.split(',')
temp = 'hot,cool,mild,hot,mild,cool,cool,mild,mild,hot,hot,mild,
   cool,mild'.split(',')
humidity = 'high,normal,high,normal,high,normal,normal,normal,high,
   high,high,high,normal,normal'.split(',')
windy = 'FALSE,TRUE,TRUE,FALSE,FALSE,FALSE,TRUE,FALSE,TRUE,FALSE,
   TRUE,FALSE,FALSE,TRUE'.split(',')
play = 'yes,yes,yes,yes,yes,yes,no,yes,no,no,no,no,yes,yes'.split('
   ,')
```

6.1.3 Creating a 'Pandas' dataframe from the dictionary

```
#creating a datadrame of input data
dataset ={'outlook':outlook,'temp':temp,'humidity':humidity,'windy'
   :windy,'play':play}
df = pd.DataFrame(dataset,columns=['outlook','temp','humidity','
   windy','play'])
```

6.1.4 Steps in LDT algorithm

Step 1: compute the entropy for data-set

Step 2: for every attribute/feature:

- calculate entropy for all categorical values
- take average information entropy for the current attribute
- calculate gain for the current attribute

Step 3: pick the highest gain attribute

Step 4: Repeat until we get the tree we desired

6.1.5 Working code for ID3 LDT algorithm-defining functions

```
# calculating entropy of the sample space
def find_entropy(df):
 Samplespace = df.keys()[-1]   #To make the code generic, changing
    target variable class name
```

```
 entropy = 0
 values = df[Samplespace].unique()
 for value in values:
   fraction = df[Samplespace].value_counts()[value]/len(df[
      Samplespace])
   entropy += -fraction*np.log2(fraction)
 return entropy
```

```
#entropy of attributes
def find_entropy_attribute(df,attribute):
  Samplespace = df.keys()[-1]   #To make the code generic, changing
      target variable class name
  target_variables = df[Samplespace].unique()  #This gives all 'Yes
     ' and 'No'
  variables = df[attribute].unique()    #This gives different
     features in that attribute (like 'Hot','Cold' in Temperature)
  entropy2 = 0
  for variable in variables:
    entropy = 0
    for target_variable in target_variables:
        num = len(df[attribute][df[attribute]==variable][df[
           Samplespace] ==target_variable])
        den = len(df[attribute][df[attribute]==variable])
        fraction = num/(den+eps)
        entropy += -fraction*log(fraction+eps)
    fraction2 = den/len(df)
    entropy2 += -fraction2*entropy
  return abs(entropy2)
```

```
def find_winner(df):
 Entropy_att = []
 IG = []
 for key in df.keys()[:-1]:
    IG.append(find_entropy(df)-find_entropy_attribute(df,key))
 return df.keys()[:-1][np.argmax(IG)]
##############################
def get_subtable(df, node,value):
   return df[df[node] == value].reset_index(drop=True)
```

```
def buildTree(df,tree=None):
  Samplespace = df.keys()[-1]   #To make the code generic, changing
      target variable class name
  DEC=df.columns[-1]
#Here we build our decision tree

#Get attribute with maximum information gain
  node = find_winner(df)

#Get distinct value of that attribute e.g Salary is node and Low,
   Med and High are values
  attValue = np.unique(df[node])

#Create an empty dictionary to create tree
```

```
  if tree is None:
     tree={}
     tree[node] = {}

#We make loop to construct a tree by calling this function
   recursively.
#In this we check if the subset is pure and stops if it is pure.

for value in attValue:

    subtable = get_subtable(df,node,value)
    clValue,counts = np.unique(subtable[DEC],return_counts=True)

    if len(counts)==1:#Checking purity of subset
       tree[node][value] = clValue[0]
    else:
        tree[node][value] = buildTree(subtable) #Calling the
           function recursively

  return tree
```

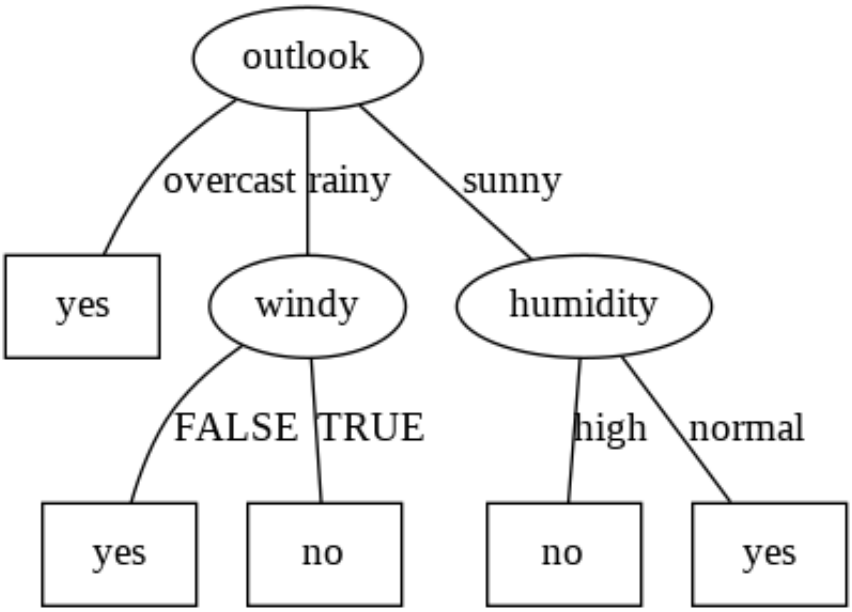
6.1.6 Running the LDT algorithm

```
#running the code to build the tree with given dataset
tree=buildTree(df)
```

6.1.7 Printing the tree structure

```
# printing the tree
import pprint
pprint.pprint(tree)
## output as a dictionary:
{'outlook': {'overcast': 'yes',
                'rainy': {'windy': {'FALSE': 'yes', 'TRUE': 'no'}},
                'sunny': {'humidity': {'high': 'no', 'normal': 'yes
                   '}}}}
```

6.1.8 The python Tree in the form of a tree diagram



6.1.9 Example 2:

Form a decision tree for the following dataset using Learning Decision Tree algorithm (KTU Model QP).

Example	A	B	C	Response
a	1	0	0	0
b	1	0	1	0
c	0	1	0	0
d	1	1	1	1
e	1	1	0	1

6.1.10 Example 3:

Form a decision tree for the following dataset using Learning Decision Tree algorithm.

Example	X	Y	Z	Response
a	1	1	1	1
b	1	1	0	1
c	0	0	1	2
d	1	0	0	2

6.1.11 Example 4:

Let's assume you want to play badminton on Saturday. Let's say you go out and check if it's hot or cold, check the speed of the wind and humidity, how the weather is, i.e. is it sunny, cloudy, or rainy. You take all these factors into account to decide if you want to play or not. A 10 day activity report based on the whether conditions is given below. Develop a decision tree based on this data

Day	Weather	Temperature	Humidity	Windy	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

References

[1] Murphy, K. P. (2022). Probabilistic machine learning: an introduction. MIT press.

[2] Stuart, J. "Artificial Intelligence A Modern Approach Third Edition." (2010).

[3] Bishop, Christopher M., and Nasser M. Nasrabadi. Pattern recognition and machine learning. Vol. 4. No. 4. New York: springer, 2006.