Machine Learning Lab

Name : Aman Kumar Gupta

Roll No.: BTech/25013/18

Branch : CSE

Semester : 6

Candidate Elimination Algorithm

Terms Used:

1. **Concept learning**: Concept learning is basically learning task of the machine (Learn by Train data)
2. **General Hypothesis**: Not Specifying features to learn the machine.
3. **G** = `{‘?’, ‘?’,’?’,’?’…}` : Number of attributes
4. **Specific Hypothesis**: Specifying features to learn machine (Specific feature)
5. **S**= `{‘pi’,’pi’,’pi’…}` : Number of pi depends on number of attributes.
6. **Version Space**: It is intermediate of general hypothesis and Specific hypothesis. It not only just written one hypothesis but a set of all possible hypothesis based on training data-set.

Algorithm:

```
Step1: Load Data set
Step2: Initialize General Hypothesis  and Specific  Hypothesis.
Step3: For each training example
Step4: If example is positive example
         if attribute_value == hypothesis_value:
            Do nothing
         else:
            replace attribute value with '?' (Basically generalizing it)
Step5: If example is Negative example
         Make generalize hypothesis more specific.
```

In [1]:
```python
#import usful packages

import numpy as np
import pandas as pd
```

In [2]:
```python
#read data set
df = pd.read_csv('data/ce.csv')
```

In [3]:
```python
#preview dataset
df
```

Out[3]:

	sky	airtemp	humidity	wind	water	forcast	enjoysport
0	sunny	warm	normal	strong	warm	same	Yes
1	sunny	warm	high	strong	warm	same	Yes
2	rainy	cold	high	strong	warm	change	No
3	sunny	warm	high	strong	cool	change	Yes

In [4]:
```python
#defining algorithm
def candidate_elimination(con,tar):
    for i,val in enumerate(tar):
        if val == 'Yes':
            specific_h = con[i].copy()
            break
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))

    for i, val in enumerate(con):
        if tar[i] == "Yes":
            for j in range(len(specific_h)):
                if val[j]!= specific_h[j]:
                    specific_h[j] ='?'
                    general_h[j][j] ='?'

        if tar[i] == "No":
            for j in range(len(specific_h)):
                if val[j]!= specific_h[j]:
                    general_h[j][j] = specific_h[j]
                else:
                    general_h[j][j] = '?'
    return specific_h,general_h
```

In [5]:
```python
#run algorithm
con = np.array(df)[:,:-1]
tar = np.array(df)[:,-1]

spe_h,gen_h = candidate_elimination(con,tar)
```

In [6]:
```
#print specific_h
spe_h
```

Out[6]:
```
array(['sunny', 'warm', '?', 'strong', '?', '?'], dtype=object)
```

In [7]:
```
#print gen_h
gen_h
```

Out[7]:
```
[['sunny', '?', '?', '?', '?', '?'],
 ['?', 'warm', '?', '?', '?', '?'],
 ['?', '?', '?', '?', '?', '?'],
 ['?', '?', '?', '?', '?', '?'],
 ['?', '?', '?', '?', '?', '?'],
 ['?', '?', '?', '?', '?', '?']]
```

Find-S algorithm:

Important Representation:

1. `?` indicates that any value is acceptable for the attribute.
2. specify a single required value (e.g., Cold) for the attribute.
3. `φ` indicates that no value is acceptable.
4. The most general hypothesis is represented by: `{?, ?, ?, ?, ?, ?}`
5. The most specific hypothesis is represented by : `{φ, φ, φ, φ, φ, φ}`

Algorithm:

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x For each attribute constraint a, in h

```
     If the constraint a, is satisfied by x
     Then do nothing
     Else replace a, in h by the next more general constraint that is satisfi
    ed by x
```

3. Output hypothesis h

In [7]:
```
#import usful packages

import numpy as np
import pandas as pd
```

In [8]:
```python
#read data set
df = pd.read_csv('data/fs.csv')
```

In [9]:
```python
#preview dataset
df
```

Out[9]:

	Color	Toughness	Fungus	Appearance	Poisonous
0	Green	Hard	No	Wrinkled	Yes
1	Green	Hard	Yes	Smooth	No
2	Brown	Soft	No	Wrinkled	No
3	Orange	Hard	No	Wrinkled	Yes
4	Green	Hard	Yes	Wrinkled	Yes
5	Orange	Hard	No	Wrinkled	Yes

In [10]:
```python
# find s algo
def find_s(con,tar):
    for i,val in enumerate(tar):
        if val == 'Yes':
            specific_h = con[i].copy()
            break

    for i,val in enumerate(con):
        if tar[i] == 'Yes':
            for j,feat in enumerate(val):
                if specific_h[j] != feat:
                    specific_h[j] = '?'
            print(specific_h)
    return specific_h
```

In [11]:
```python
#run algorithm
con = np.array(df)[:,:-1]
tar = np.array(df)[:,-1]

gen_h = find_s(con,tar)
```

```
['Green' 'Hard' 'No' 'Wrinkled']
['?' 'Hard' 'No' 'Wrinkled']
['?' 'Hard' '?' 'Wrinkled']
['?' 'Hard' '?' 'Wrinkled']
```

In [12]:
```python
#see result

gen_h
```

Out[12]:
```
array(['?', 'Hard', '?', 'Wrinkled'], dtype=object)
```

A. Simple Linear Analysis (From Scratch)

In [1]:
```
#import package
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
#read data
df = pd.read_csv('data/a.csv')
df.head()
```

Out[2]:

	X	Y
0	3.5	5.1
1	3.0	4.9
2	3.2	4.7
3	3.1	4.6
4	3.6	5.0

Suppose the required regression line equation is :

$$y = m.x + b$$

There are two equations called normal equation from which we can find `m` and `b` :

$$\Sigma XY = b.\Sigma X + m.\Sigma X^2 \qquad \ldots\ldots\ldots\ldots(1)$$
$$\Sigma Y = n.b + m.\Sigma X \qquad \ldots\ldots\ldots\ldots(2)$$

Or we can solve the two equation for m and b as:

$$m = \frac{n.\Sigma xy - \Sigma x.\Sigma y}{n.\Sigma x^2 - (\Sigma x)^2}$$
$$b = \overline{y} - m.\overline{x}$$

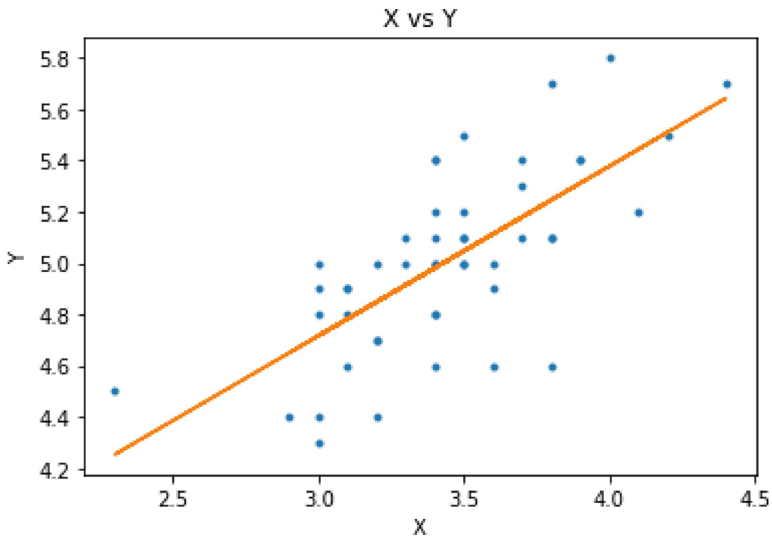
In [3]:
```
#Calculate m and b for y = mx+b
X = np.array(df['X'])
Y = np.array(df['Y'])

m = ((len(X)*sum(X*Y)) - sum(X)*sum(Y))/((len(X)*sum(X**2)) - (sum(X)**2))

b = np.mean(Y)- m*np.mean(X)
```

In [4]:
```
# append calculated y column in df
df['Y_h'] = m*X + b
```

In [5]:
```
#plot data points and y=mx+b line
plt.plot(df['X'],df['Y'],'.')
plt.plot(df['X'],df['Y_h'],'-')
plt.xlabel("X");
plt.ylabel("Y");
plt.title("X vs Y");
```



B. Multiple Linear Regression (From Scratch)

In [6]:
```
df1 = pd.read_csv("data/mlr.csv")
df1.head()
```

Out[6]:

	X1	X2	X3	X4	X5
0	6.8	225	0.442	0.672	9.2
1	6.3	180	0.435	0.797	11.7
2	6.4	190	0.456	0.761	15.8
3	6.2	180	0.416	0.651	8.6
4	6.9	205	0.449	0.900	23.2

Data Description

The following data (X1, X2, X3, X4, X5) are for each player.

X1 = height in feet

X2 = weight in pounds

X3 = percent of successful field goals (out of 100 attempted)

X4 = percent of successful free throws (out of 100 attempted)

X5 = average points scored per game

In [7]:
```
X = np.array(df1.drop('X5',axis = 1))
ones = np.ones((X.shape[0],1))
X = np.append(ones,X,axis = 1)
Y = np.array(df1['X5'])
```

$$y = b_0 + x_1 b_1 + x_2 b_2 + \cdots$$

$$Y = Xb$$

$$\mathbf{X} = \begin{bmatrix} 1 & X_1 & X_2 & \cdots \\ 1 & \cdots & \cdots & . \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \cdots & \cdots & \cdots \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} b_0 & b_1 & b_2 \cdots \end{bmatrix}$$

In [8]:
```
b = np.linalg.inv(X.T@X)@X.T@Y
b
```

Out[8]:
```
array([ 4.14870671e+00, -3.69049908e+00,  9.45845788e-03,  4.79401992e+01,
        1.13710193e+01])
```

In [9]:
```
Y_pred = X@b
Y_pred
```

Out[9]:
```
array([10.01235894, 12.51777389, 12.84069605, 10.3157912 , 12.38231366,
       12.18928646, 14.76231003, 12.2882684 , 11.06248103, 13.80102549,
       12.92521215, 13.24396498,  8.64006433, 12.60525851, 12.92430257,
       15.2545908 , 15.58138272,  5.09650334,  9.94940424, 12.49696124,
        9.05637536, 10.25083143,  8.06191203, 10.07255781, 14.74963484,
       12.07610898, 13.95300287,  8.71473239, 11.3419041 , 14.40707455,
       15.27283126, 10.4123043 , 13.64991307, 10.25083143,  8.97445925,
       14.70763753, 14.90726563, 13.253166  ,  7.32888666,  6.6597904 ,
        8.95334041,  5.83073166, 13.82675288, 16.36627839, 12.00928465,
       10.11871685, 17.63471509,  9.70871125,  8.95738083, 14.73644788,
       15.93275941, 12.25972646, 11.34188054, 10.03210379])
```

$$R^2 = \frac{\Sigma(\hat{Y} - \overline{Y})^2}{\Sigma(Y - \overline{Y})^2}$$

In [10]:
```
Y_mean = np.mean(Y)

R2 = (np.sum((Y_pred-Y_mean)**2))/(np.sum((Y-Y_mean)**2))
print(f"R^2 value is {R2}")
```

```
R^2 value is 0.22225063130014133
```

Linear Regression Using SKLearn

1. Read Data

In [11]:
```
df_3 = pd.read_csv("data/insurance.csv")
df_3.head()
```

Out[11]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	3	16884.92400
1	18	1	33.770	1	0	2	1725.55230
2	28	1	33.000	3	0	2	4449.46200
3	33	1	22.705	0	0	1	21984.47061
4	32	1	28.880	0	0	1	3866.85520

In [12]:
```
df_3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 348 entries, 0 to 347
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       348 non-null    int64  
 1   sex       348 non-null    int64  
 2   bmi       348 non-null    float64
 3   children  348 non-null    int64  
 4   smoker    348 non-null    int64  
 5   region    348 non-null    int64  
 6   charges   348 non-null    float64
dtypes: float64(2), int64(5)
memory usage: 19.2 KB
```

2. Import Linear Regressor and Split Data Set

In [13]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

In [14]:
```python
X = df_3.drop('charges', axis = 1)
Y = df_3['charges']
```

In [15]:
```python
X_train,X_test,Y_train,Y_test = train_test_split(X,Y, test_size = 0.3)
```

3. Train Linear Regrression Model

In [16]:
```python
lr_model = LinearRegression()
```

In [17]:
```python
lr_model.fit(X_train,Y_train)
```

Out[17]:
```
LinearRegression()
```

4. Predict and check Model

In [18]:
```python
loc = 10
x = X_test.iloc[loc]
y = Y_test.iloc[loc]
y_pred = lr_model.predict([x])
```

In [19]:
```python
print(f'x = {x.values}\ny = {y}\nPredicted Value = {y_pred[0]}')
```

```
x = [19.   0.   28.9  0.   0.   3. ]
y = 1743.214
Predicted Value = 1913.0976139823597
```

In [20]:
```python
print(f'Score of Model = {lr_model.score(X_test,Y_test)}')
```

```
Score of Model = 0.7750219073416568
```

Locally Weight Regression (From Scretch)

In [1]:
```python
# import useful module
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Algorithm

Locally weighted Regression is a non parametric Regression Algorithms in which we need data every time we calculate matrix.

Cost Function in Linear Regression :

$$\mathbf{costfn} = \Sigma(y_i - \theta x_i)^2$$

Cost Function for Locally Weighted Regression:

$$\mathbf{costfn} = \Sigma w_i(y_i - \theta x_i)^2$$

where x_i, y_i is the $i^t h$ example and w_i is weight.
Generally

$$w_i = e^{-\frac{(x_i - x)^2}{2\tau^2}}$$

by minimize `costfn` for θ and solving expression we get

$$\theta = (X^T W X)^{-1} (X^T W Y)$$

And prediction is $y = X\theta$.

In [2]:
```
def w_i(point,X,tau):
    m,n = X.shape
    w = np.mat(np.eye(m))

    for j in range(m):
        diff = X[j]-point
        w[j, j] = np.exp(diff * diff.T / (-2.0 * tau**2))
    return w
```

In [3]:
```
def theta(X,Y,point,tau):
    W = w_i(point,X,tau)
    th_x = X.T@W@X
    th_y = X.T@W@Y.T
    th = th_x.I@th_y
    return th
```

In [4]:
```
def predict(X,Y,tau):
    m,n = np.shape(X)
    ypred = np.zeros(m)

    for i in range(m):
        ypred[i] = X[i] * theta( X, Y, X[i],tau)
    return ypred
```

Part 1

Generate X and Y and apply

In [5]:
```
n = 100
xs = np.linspace(0, np.pi, n)
ys = 1 + np.sin(xs) + np.cos(xs**2) + np.random.normal(0, 0.1, n)
```

In [6]:
```
X = np.mat(xs)
Y = np.mat(ys)

#add 1 with each entry in X
m = X.shape[1]
one = np.ones((1,m))
X = np.hstack((one.T,X.T))

print(X[:5])
```

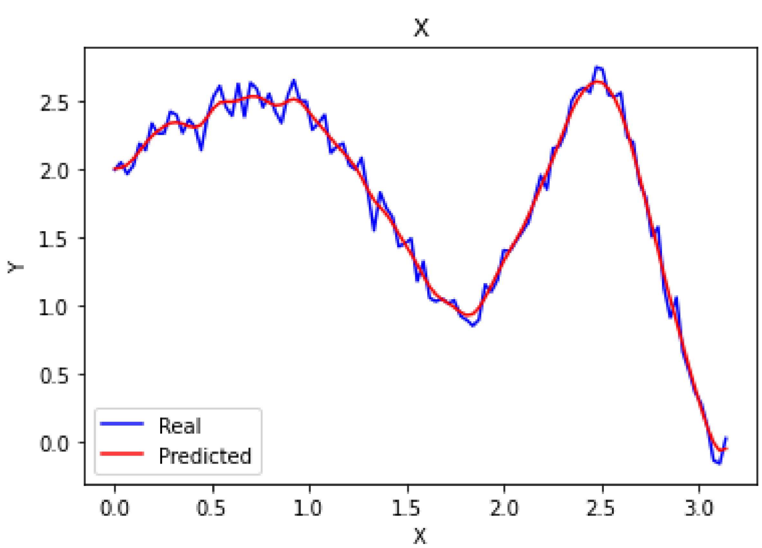
```
[[1.         0.        ]
 [1.         0.03173326]
 [1.         0.06346652]
 [1.         0.09519978]
 [1.         0.12693304]]
```

In [7]:
```
Y_prediction_05 = predict(X,Y,0.05)
```

In [8]:
```
X_l = xs
Y_l = ys
Y_p = Y_prediction_05

plt.plot(X_l,Y_l,'b-')
plt.plot(X_l,Y_p,'r-')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('X')
plt.legend(['Real','Predicted'])
```

Out[8]:
```
<matplotlib.legend.Legend at 0x2202eeed070>
```



Part 2

Use on DataSet

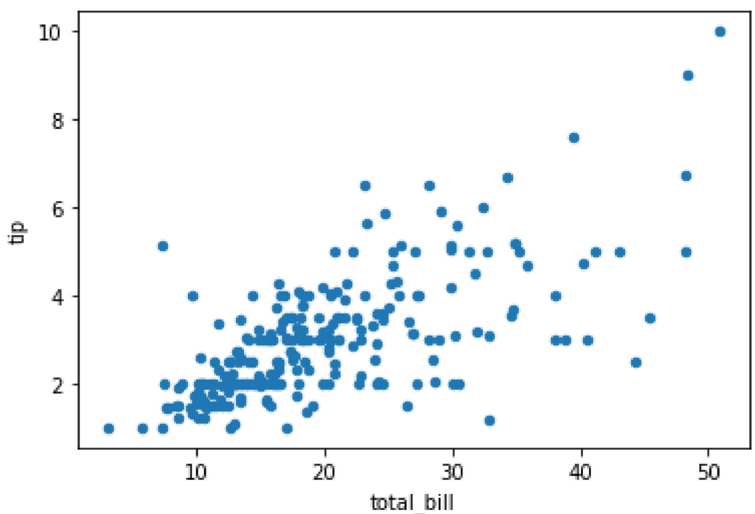
In [9]:
```
# Read Data and make DataFrame
df = pd.read_csv('data/tips.csv')
df.head()
```

Out[9]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

In [10]:
```
#plot total_bill vs tip
df.plot(x = 'total_bill',y = 'tip',kind = 'scatter')
```

Out[10]: <AxesSubplot:xlabel='total_bill', ylabel='tip'>



In [11]:
```
# select columns regression
X = np.mat(df['total_bill'])
Y = np.mat(df['tip'])

#add 1 with each entry in X
m = X.shape[1]
one = np.ones((1,m))
X = np.hstack((one.T,X.T))

print(X[:5])
```

```
[[ 1.   16.99]
 [ 1.   10.34]
 [ 1.   21.01]
 [ 1.   23.68]
 [ 1.   24.59]]
```

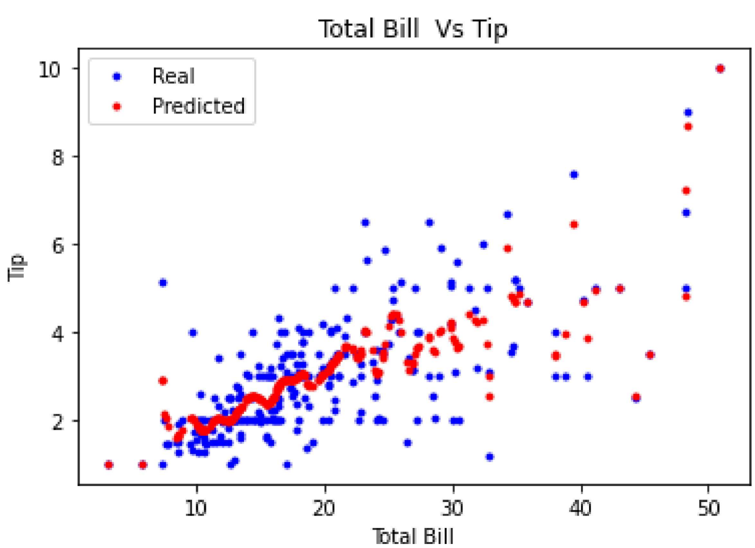
In [12]:
```
Y_prediction = predict(X,Y,0.4)
```

Plot the predicted result

In [13]:
```
X_l = df['total_bill']
Y_l = df['tip']
Y_p = Y_prediction

plt.plot(X_l,Y_l,'b.')
plt.plot(X_l,Y_p,'r.')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.title('Total Bill  Vs Tip')
plt.legend(['Real','Predicted'])
```

Out[13]:
```
<matplotlib.legend.Legend at 0x2202f37ef70>
```



Decision Tree

In [1]:
```
#import useful package
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix,accuracy_score
```

In [2]:
```
#read data
df = pd.read_csv('data/Iris.csv')
df.head()
```

Out[2]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [4]:
```
#Splitting data in to train and test part

X = df.drop(['Species','Id'],axis=1)
class_name = list(df.Species.unique())
feat_name = list(X.columns)
Y = df.Species.apply(lambda x: class_name.index(x))
#Y = df.Species
```

In [5]:
```
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size = 0.3)
```

Train Model :- ID3

Algorithm

```
ID3 (Examples, Target_Attribute, Attributes)
    Create a root node for the tree
    If all examples are positive, Return the single-node tree Root, with labe
l = +.
    If all examples are negative, Return the single-node tree Root, with labe
l = -.
    If number of predicting attributes is empty, then Return the single node
 tree Root,
    with label = most common value of the target attribute in the examples.
    Otherwise Begin
        A ← The Attribute that best classifies examples.
        Decision Tree attribute for Root = A.
        For each possible value, vi, of A,
            Add a new tree branch below Root, corresponding to the test A = v
i.
            Let Examples(vi) be the subset of examples that have the value vi
for A
            If Examples(vi) is empty
                Then below this new branch add a leaf node with label = most
 common target value in the examples
            Else below this new branch add the subtree ID3 (Examples(vi), Tar
get_Attribute, Attributes – {A})
    End
    Return Root
```

ID3 algorithm is based on entropy and information gain calculation.
Entropy is calculated as

$$\textbf{Entropy} = -\Sigma p(X) \log p(X)$$

Where `p(X)` is a Fraction of example in given class

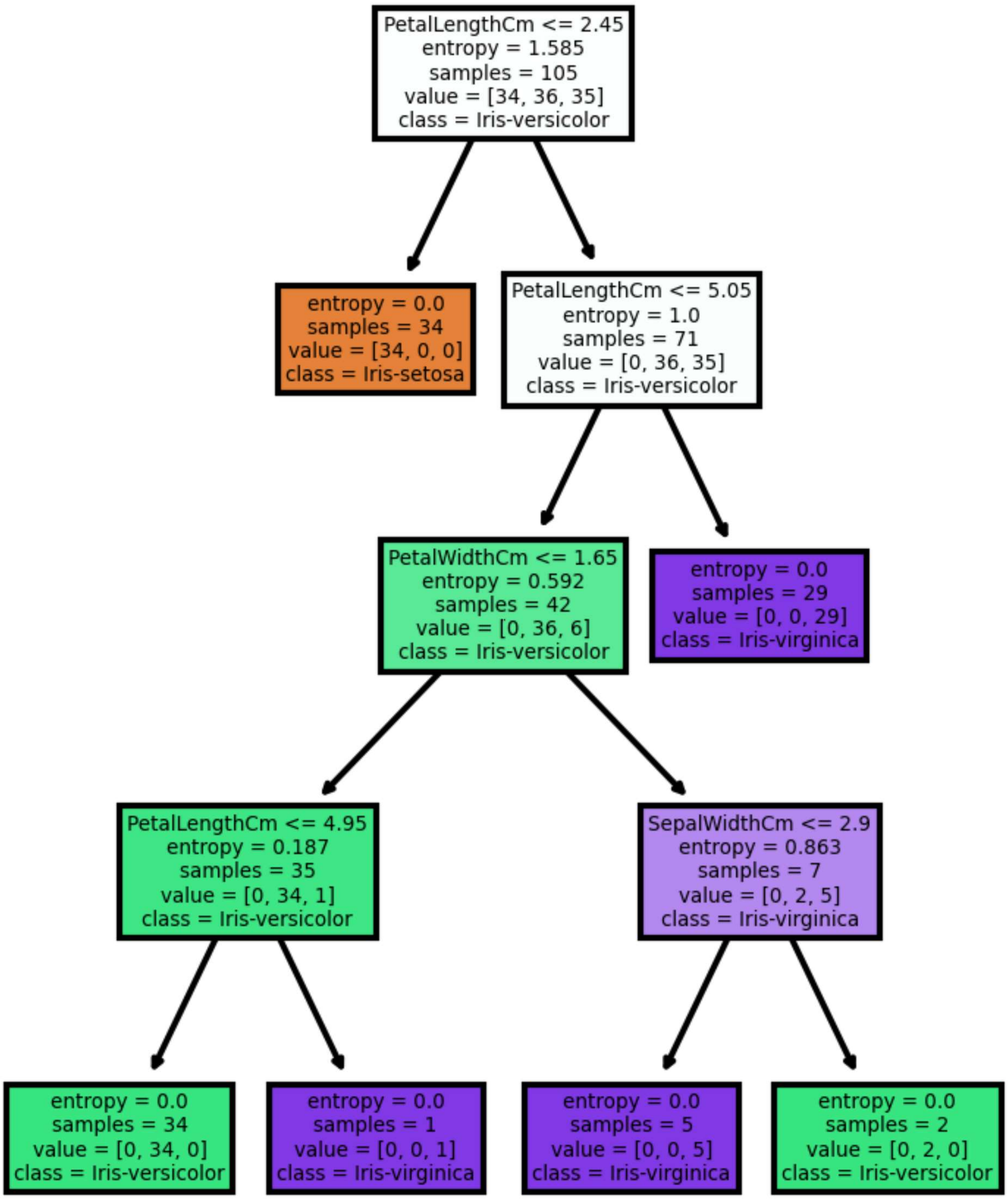
and information gain is calculated as-

$$\textbf{GAIN} = \textbf{Entropy(p)} - \Sigma \frac{n_i}{n} \textbf{Entropy(i)}$$

In [6]:
```python
cd3 = DecisionTreeClassifier(criterion='entropy')
cd3.fit(x_train,y_train)
```

Out[6]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [15]:
```
#plot tree
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (3,4), dpi=300)
plot_tree(cd3,
         feature_names = feat_name,
          class_names=class_name,
          filled = True);
```



In [8]:
```
#check accurecy
y_pred = cd3.predict(x_test)
print(f'Accuracy Score of model is : {accuracy_score(y_test,y_pred)}')
```

```
Accuracy Score of model is : 0.9333333333333333
```

In [9]:
```
print('Confusion Metrics :')
print(confusion_matrix(y_test,y_pred))
```

```
Confusion Metrics :
[[16  0  0]
 [ 0 13  1]
 [ 0  2 13]]
```

Train Model :- CART

Algorithm

```
Step 1: Start at the root node with all training instances
Step 2: Select an attribute on the basis of splitting criteria
Step 3: Partition instances according to selected attribute recursively
```

ID3 algorithm is based on GINI Impurity.
GINI is calculated as

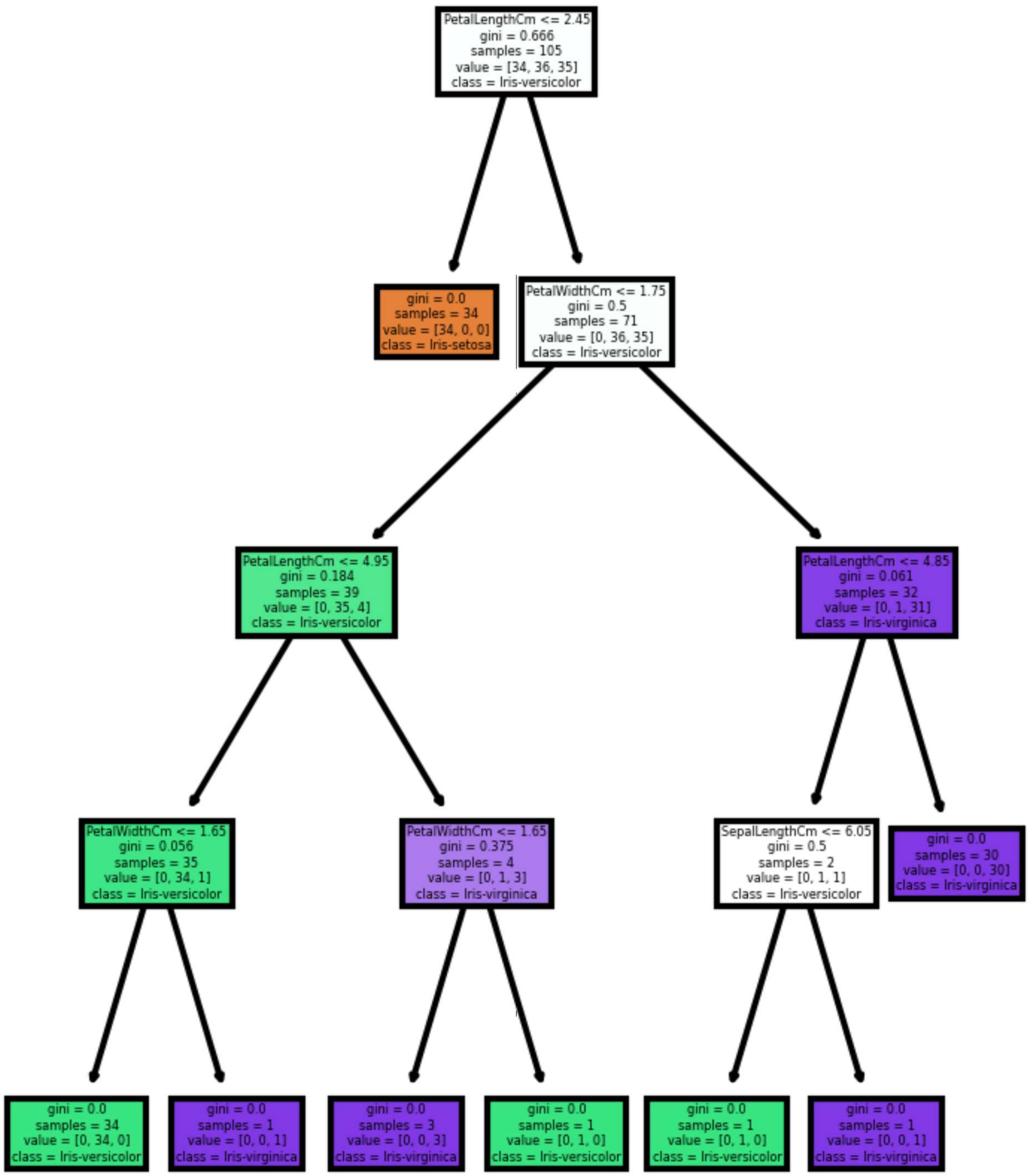
$$\textbf{GINI} = 1 - \Sigma(p_i)^2$$

Where p_i is the probability that a tuple in `D` belongs to the class `C`

In [10]:
```
cart = DecisionTreeClassifier(criterion='gini')
cart.fit(x_train,y_train)
```

Out[10]:
```
DecisionTreeClassifier()
```

In [14]:
```python
#plot tree
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (3,4), dpi=300)
plot_tree(cart,
         feature_names = feat_name,
          class_names=class_name,
          filled = True);
```



In [12]:
```
#check accurecy
y_pred = cart.predict(x_test)
print(f'Accuracy Score of model is : {accuracy_score(y_test,y_pred)}')
```

```
Accuracy Score of model is : 0.9555555555555556
```

In [13]:
```
print('Confusion Metrics :')
print(confusion_matrix(y_test,y_pred))
```

```
Confusion Metrics :
[[16  0  0]
 [ 0 13  1]
 [ 0  1 14]]
```

Logestic Regression

Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability.

For Logestic Regression Output Function :

$$\sigma(z) = \textbf{sigmoid}(z)$$

Where

$$z = \beta_0 + \beta_1 x + \cdots$$

And

$$sigmoid(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-\beta_0 + \beta_1 x + \cdots}}$$

$\sigma(z)$ gives values between 0 and 1.

and Cost Function is :

$$J(\theta) = -\frac{1}{m}\Sigma\left[y \log \sigma(z) + (1 - y) \log (1 - \sigma(z))\right]$$

To minimise our cost Function we use `Gradient Descent` .

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (\sigma(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

In [10]:
```
#import useful package
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix,accuracy_score
```

In [11]:
```
df = pd.read_csv('data/Social_Network_Ads.csv')
df.head()
```

Out[11]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   User ID          400 non-null    int64 
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64 
 3   EstimatedSalary  400 non-null    int64 
 4   Purchased        400 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

In [13]:
```
#convert class feature in numerical
g_list = ['Male','Female']
df.Gender = df.Gender.apply(lambda x: g_list.index(x))
```

In [15]:
```
#Splitting Data in train and test
X = df.drop(['User ID','Purchased'],axis = 1)
Y = df['Purchased']

x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size = 0.3)
```

In [16]:
```
#Training Logestic Regression Model
LogReg = LogisticRegression()
LogReg.fit(X,Y)
```

Out[16]:
```
LogisticRegression()
```

In [26]:
```
print("Coefficient for Regression:")
print(*LogReg.coef_[0],sep = '\n')
```

```
Coefficient for Regression:
-9.322295567048426e-11
-2.1041517809748262e-09
-2.693014040541572e-06
```

In [28]:
```
#Test accurecy
y_pred = LogReg.predict(x_test)

print(f'Accurecy = {accuracy_score(y_test,y_pred)}')
```

```
Accurecy = 0.6416666666666667
```