MCA 304
Machine Learning Lab File

Mohammad Muzammil Khan [1]

Dept. of Computer Science and Engineering,
School of Engineering Sciences and Technology,
Jamia Hamdard - India

[1]Contact: mmkhan.sch@jamiahamdard.ac.in

Enrollment no:	2020-501-029
Course:	Master of Computer Application
Year/Semester:	II/III (Class of '22)

Contents

Question 1

Find-S Algorithm

1.1 Introduction

The find-S algorithm is a basic concept learning algorithm in machine learning. The find-S algorithm finds the most specific hypothesis that fits all the positive examples.

1.2 Algorithm

1. Start with the most specific hypothesis.
2. h = $\{\phi,\phi,\phi,\phi,\phi,\phi,\phi\}$
3. Take the next example and if it is negative, then no changes occur to the hypothesis.
4. If the example is positive and we find that our initial hypothesis is too specific then we update our current hypothesis to a general condition.
5. Keep repeating the above steps till all the training examples are complete.
6. After we have completed all the training examples we will have the final hypothesis when can use to classify the new examples.

1.3 Implementation

[1]:
```
import csv
```

[2]:
```
with open('./data/weather2.csv', 'r') as f:
    reader = csv.reader(f)
    data = list(reader)
data
```

[2]:
```
[['Sky', 'AirTemp', 'Humidity', 'Wind', 'Water', 'Forecast', 'EnjoySport'],
 ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'],
 ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'],
 ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'],
 ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']]
```

[3]:
```
output_attr = (-1, "Yes", "No")
```

[4]:
```
data
```

[4]:
```
[['Sky', 'AirTemp', 'Humidity', 'Wind', 'Water', 'Forecast', 'EnjoySport'],
 ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'],
 ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'],
 ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'],
 ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']]
```

[5]:
```
S = [None] * (len(data[0]) - 1)
S
```

[5]:
```
[None, None, None, None, None, None]
```

[6]:
```
for i in data[1:]:
    if i[output_attr[0]] != output_attr[1]:
        continue
    print(i)
    j = 0
    for k in i:
        if k == output_attr[1]:
            continue
        if k != S[j]:
            if S[j] == None:
                S[j] = k
            else:
                S[j] = '?'
        j = j + 1
```

```
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
```

[7]:
```
print("Specific hypothesis is", S)
```

```
Specific hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Question 2

Linear Regression

2.1 Introduction

Linear regression is a linear approach for modelling the relationship between a dependent and independent variable. Linear regression uses representation in a linear equation that combines a specific set of input values (x) the solution to which is the predicted output for that set of input values (y). As such, both the input values (x) and the output value are numeric. The linear equation assigns one scale factor to each input value or column, called a coefficient and one additional coefficient is also added, giving the line an additional degree of freedom for moving up and down on a two-dimensional plot and is often called the intercept or the bias coefficient.

$$\hat{y} = \beta_0 + \beta_1 x$$

2.2 Steps for Calculation

Step 1: Calculate mean of x and y represented as x' and y'.

Step 2: Calculate deviation from mean for x and y with $(x - x')$ and $(y - y')$.

Step 3: Square the deviation as $(x - x')^2$.

Step 4: Calculate $(x - x')(y - y')$.

Step 5: $\beta_0 = \dfrac{\sum((x - x')(y - y'))}{\sum(x - x')^2}$

Step 6: $\beta_1 = y' - c * x'$

Step 7: $\hat{y} = \beta_0 + \beta_1 x$

where, $\hat{y}$ = dependent, β_0 = intersection, β_1 tangent, x = independent

2.3 Implementation from scratch

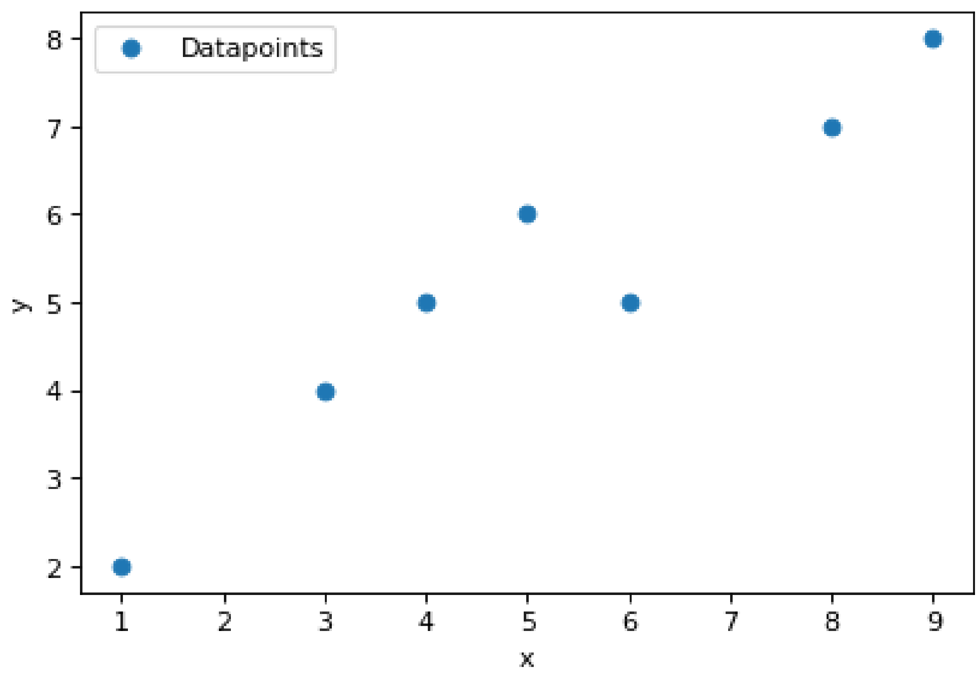
[1]:
```
import numpy as np
from matplotlib import pyplot as plt
```

[2]:
```
def coeff(x, y):
    n = np.size(x)
```

```
    print("Number of datapoints:", n)

    mean_x = np.mean(x)
    mean_y = np.mean(y)

    print("Means (x, y): ", (mean_x, mean_y))

    # cross deviation calculation
    s_xy = np.sum(y * x) - n * mean_y * mean_x;
    s_xx = np.sum(x * x) - n * mean_x * mean_x;

    b1 = s_xy/s_xx
    b0 = mean_y - b1 * mean_x

    print("Found b0 and b1: ", b0, b1)
    return (b0, b1)
```

[3]:
```
x = np.array([1, 3, 4, 5, 6, 8, 9])
y = np.array([2, 4, 5, 6, 5, 7, 8])

plt.figure(dpi=80)
plt.scatter(x, y, label="Datapoints")
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

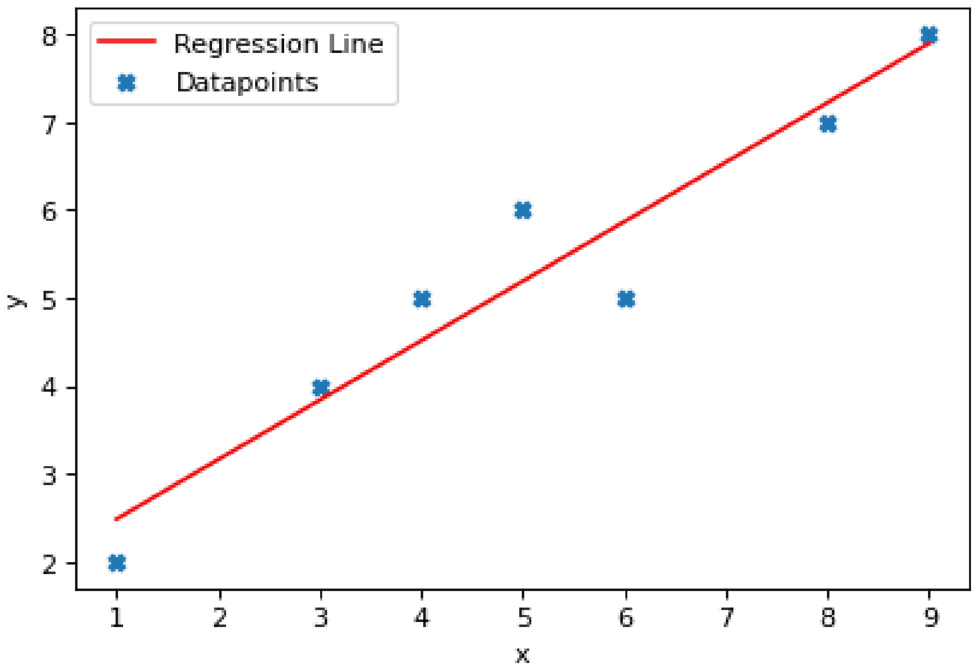
[3]: <matplotlib.legend.Legend at 0x1b7faf83bb0>



[4]:
```
b = coeff(x, y)
```

```
Number of datapoints: 7
Means (x, y):  (5.142857142857143, 5.285714285714286)
Found b0 and b1:  1.8048780487804876 0.6768292682926829
```

[5]:
```
y_pred = b[0] + b[1] * x

plt.figure(dpi=80)
plt.scatter(x, y, marker="X", label='Datapoints')
plt.plot(x, y_pred, color="red", label='Regression Line')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



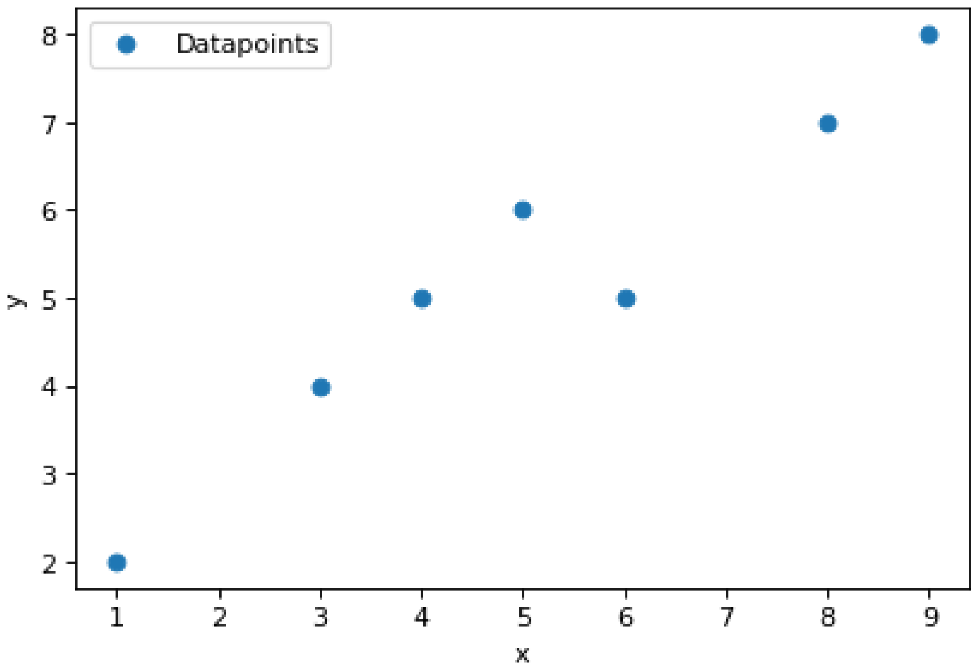
2.4 Implementation using scikit-learn

[6]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing, svm
from sklearn.linear_model import LinearRegression
```

[7]:
```
x = pd.DataFrame([1, 3, 4, 5, 6, 8, 9])
y = pd.DataFrame([2, 4, 5, 6, 5, 7, 8])
```

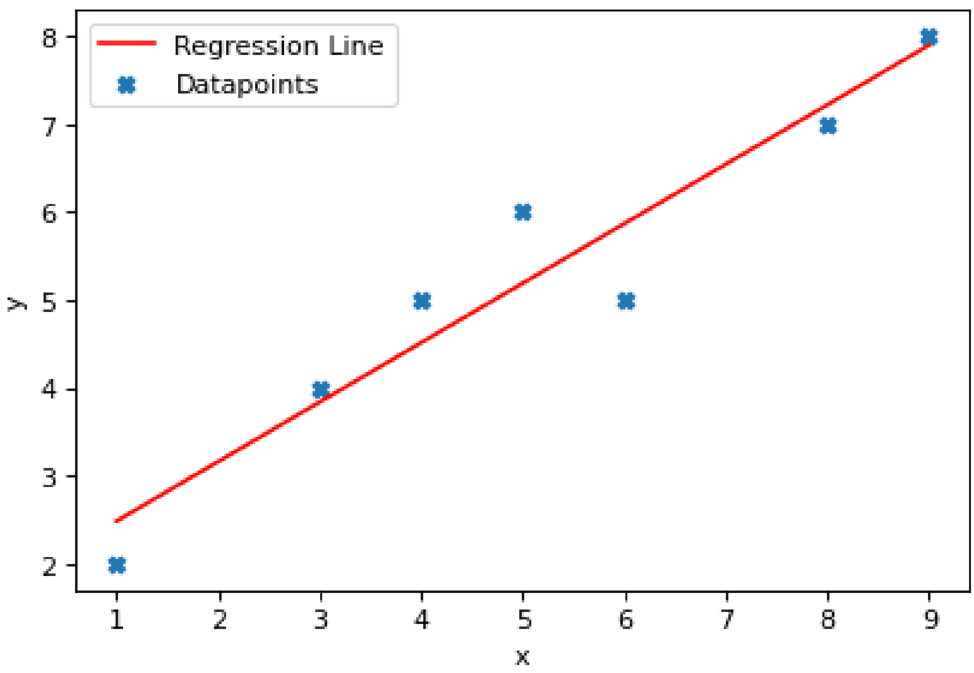
```
plt.figure(dpi=80)
plt.scatter(x, y, label="Datapoints")
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

[7]: <matplotlib.legend.Legend at 0x1b7817b93d0>



[8]:
```
regr = LinearRegression()
regr.fit(x, y)
y_pred = regr.predict(x)
```

[9]:
```
plt.figure(dpi=80)
plt.scatter(x.values, y.values, marker="X", label='Datapoints')
plt.plot(x.values, y_pred, color="red", label='Regression Line')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```


Regression Line
Datapoints
y
x
8
7
6
5
4
3
2
1
2
3
4
5
6
7
8
9

Question 3

Multiple Regression

3.1 Introduction

Multiple regression is like linear regression, but with more than one independent value, meaning that we try to predict a value based on two or more variables. Multiple regression is a statistical technique that can be used to analyze the relationship between a single dependent variable and several independent variables. The objective of multiple regression analysis is to use the independent variables whose values are known to predict the value of the single dependent value. Each predictor value is weighed, the weights denoting their relative contribution to the overall prediction.

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$$

Here $\hat{y}$ is the dependent variable, and $x_1, x_2, ..., x_n$ are the n independent variables. In calculating the weights, β_0, $x_1, x_2, ..., x_n$, regression analysis ensures maximal prediction of the dependent variable from the set of independent variables. This is usually done by least squares estimation.

3.2 Implementation

[1]:
```
import pandas as pd
from sklearn import linear_model
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
dataset = pd.read_csv("./data/house_data.csv")
dataset.head()
```

[2]:

	sqft	bedrooms	price
0	1180	3	3540000
1	2570	3	7710000
2	770	2	1540000
3	1960	4	7840000
4	1680	3	5040000

[3]:
```
X = dataset.iloc[:,0:-1]
y = dataset.iloc[:,-1]
dataset.corr()
```

[3]:

	sqft	bedrooms	price
sqft	1.000000	0.410243	0.945462
bedrooms	0.410243	1.000000	0.654177
price	0.945462	0.654177	1.000000

[4]:
```
regr = linear_model.LinearRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25,␣
↪random_state=42)
regr.fit(X_train, y_train)
regr.score(X_test, y_test)
```

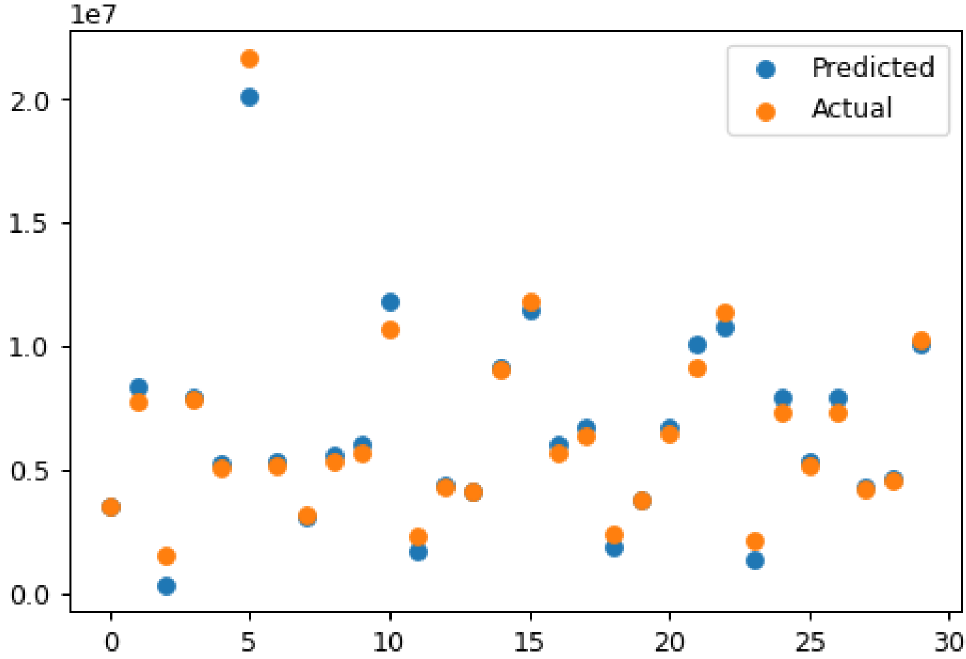
[4]: 0.976264543946519

[5]:
```
y_pred=regr.predict(X)
print("Predicted:\t", y_pred[:10])
print("Actual:\t\t", y.values[:10])
```

```
Predicted:       [ 3488660.82843106  8375217.66544624   329918.68771236
7948141.12000992
  5246415.08635019 20111800.5848103   5369457.88440453  3066799.80653047
  5597965.93793402  5984671.87467623]
Actual:          [ 3540000  7710000  1540000  7840000  5040000 21680000 ␣
↪5145000
3180000
  5340000  5670000]
```

[6]:
```
plt.figure(dpi=90)
plt.scatter(range(y_pred.size), y_pred, label="Predicted")
plt.scatter(range(y_pred.size), y, label="Actual")
plt.legend()
```

[6]: <matplotlib.legend.Legend at 0x1dc847c2af0>


1e7
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
30
Predicted
Actual

Question 4

Decision Tree

4.1 Introduction

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

4.2 Algorithm

The core algorithm for building decision trees called ID3 by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking. ID3 uses Entropy and Information Gain to construct a decision tree.

4.2.1 Entropy

A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogeneous). ID3 algorithm uses entropy to calculate the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one.

4.2.2 Information Gain

The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches).

4.3 Building a Decision tree

To build a decision tree, we need to calculate two types of entropy using frequency tables as follows:

Step 1: Calculate Entropy using the frequency table of target attribute:

$$E(S) = \sum_{i=1}^{c} -p_i log_2 p_i$$

Step 2: Calculate Entropy of attributes with respect to target attribute:

$$E(T, X) = \sum_{c \in X} P(c)E(c)$$

**P(c) probability of c*

Step 3: Calculate information gain of attributes and select the highest node as root node:

$$Gain(T, X) = E(T) - E(T, X)$$

Step 4: Generate sub-tables for attributes with respect to parent node and target node.

Step 5: Repeat untill Entropy reaches 0.

4.4 Implementation

[1]:
```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn import metrics
import matplotlib.pyplot as plt
```

[2]:
```python
dataset = pd.read_csv("./data/weather.csv")
map_dict = {"Sunny":0, "Overcast":1, "Rain":3, "Hot":0, "Mild":1, "Cool":2,␣
↪"High":0, "Normal":1, "Weak":0, "Strong":1, "Yes":1, "No":0}
dataset.head()
```

[2]:

	Outlook	Temp	Humidity	Wind	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes

[3]:
```python
X_raw = dataset.iloc[:,0:-1]
y = dataset.iloc[:,-1]
```

[4]:
```python
X = pd.DataFrame()
for x in X_raw:
    X[x] = X_raw[x].map(map_dict)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25,␣
↪random_state=42)
X
```

[4]:

	Outlook	Temp	Humidity	Wind
0	0	0	0	0
1	0	0	0	1
2	1	0	0	0
3	3	1	0	0
4	3	2	1	0
5	3	2	1	1

6	1	2	1	1
7	0	1	0	0
8	0	2	1	0
9	3	1	1	0
10	0	1	1	1
11	1	1	0	1
12	1	0	1	0
13	3	1	0	1

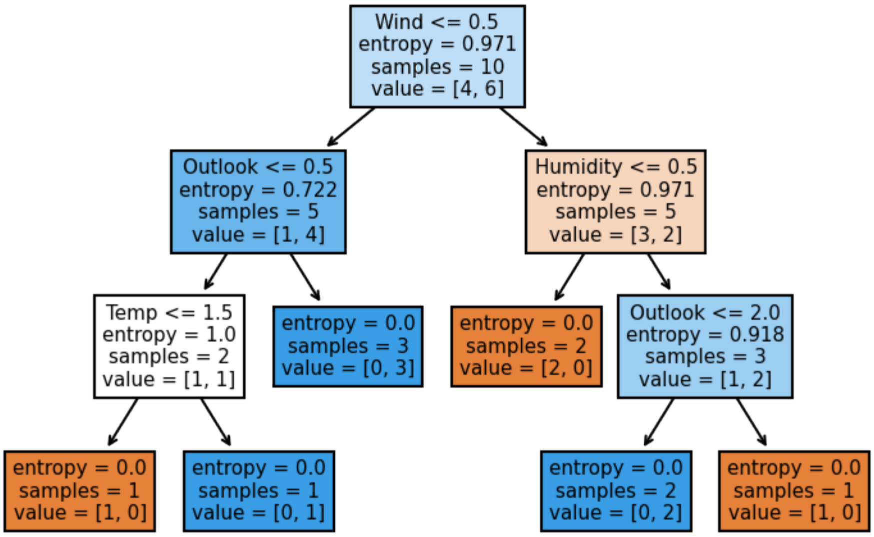
[5]:
```
model = DecisionTreeClassifier(criterion="entropy")
# Train Decision Tree Classifier
model = model.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = model.predict(X_test)
```

[6]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.75
```

[7]:
```
plt.figure(dpi=150)
plot_tree(model, feature_names=X_raw.columns, filled=True)
```

[7]:
```
[Text(348.75, 396.375, 'Wind <= 0.5\nentropy = 0.971\nsamples = 10\nvalue =␣
 ↪[4,
6]'),
 Text(209.25, 283.125, 'Outlook <= 0.5\nentropy = 0.722\nsamples = 5\nvalue =
[1, 4]'),
 Text(139.5, 169.875, 'Temp <= 1.5\nentropy = 1.0\nsamples = 2\nvalue = [1,
1]'),
 Text(69.75, 56.625, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(209.25, 56.625, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(279.0, 169.875, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(488.25, 283.125, 'Humidity <= 0.5\nentropy = 0.971\nsamples = 5\nvalue␣
 ↪=
[3, 2]'),
 Text(418.5, 169.875, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(558.0, 169.875, 'Outlook <= 2.0\nentropy = 0.918\nsamples = 3\nvalue =␣
 ↪[1,
2]'),
 Text(488.25, 56.625, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(627.75, 56.625, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]')]
```



[8]:
```
def make_prediction(case):
    df = pd.DataFrame(case)[0].map(map_dict)
    return model.predict([df])
```

[9]:
```
print(make_prediction(["Overcast","Hot","High","Strong"]))
```

```
['No']
```

[10]:
```
print(make_prediction(["Overcast","Hot","Normal","Weak"]))
```

```
['Yes']
```

Question 5

Naïve Bayes Classification

Naïve Bayes classification algorithm is based on Bayes' Theorem. The dataset is divided into two parts, namely, feature matrix and the response vector.

5.1 Bayes' Theorem

Bayes' Theorem provides a way that we can calculate the probability of a piece of data belonging to a given class, given our prior knowledge. Bayes' Theorem is stated as:

$$P(\alpha|\beta) = \frac{P(\beta|\alpha) * P(\alpha)}{P(\beta)}$$

Naive Bayes is a classification algorithm for binary (two-class) and multiclass classification problems. It is called Naive Bayes or idiot Bayes because the calculations of the probabilities for each class are simplified to make their calculations tractable.

5.2 Bayes' Theorem Calculation

In our example, we shall use `sklearn.naive_bayes` to classify on the basis of previous data wheater or not a person has gotten a flu or not. We shall also check the accuracy and generate confusion matrix of the model. However, to gain a general understanding, here is how it works:

Step 1: Study the dataset Our dataset include 4 feature columns and 1 target column.

```
Chills  Runny_nose  Headache    Fever  |  Flu
```

Step 2: Calculate probability of target

$$P(Flu|Y) = 10/14 = 0.714285714$$

$$P(Flu|N) = 4/14 = 0.285714286$$

Step 3: Calculate probability of feature columns for each

For each column of feature, calculate probability of all cases features X target. Such as -

$$P(Chills = Y|Flu = Y) = 6/10 = .6$$

$$P(Chills = Y|Flu = N) = 1/4 = .25$$

And so on...

Step 4: Calculate for the given case

$$P(\alpha|\beta) = \frac{P(\beta|\alpha) * P(\alpha)}{P(\beta)}$$

Here,

1. α, β = Event
2. $P(\alpha)$, $P(\beta)$ = Probability of event occurring
3. $P(\alpha|\beta)$ = Probability of α happening such that β is true

5.3 Introduction

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
```

[2]:
```
dataset = pd.read_csv('./data/flu.csv')
```

[3]:
```
X_raw = dataset.iloc[:,0:-1]
y = dataset.iloc[:,-1]
map_dict = {"Y": 1, "N": 0, "No": 0, "Mild": 1, "Strong": 2}
```

[4]:
```
dataset
```

[4]:

	Chills	Runny_nose	Headache	Fever	Flu
0	Y	N	Mild	Y	N
1	Y	Y	No	N	Y
2	Y	N	Strong	Y	Y
3	N	Y	Mild	Y	Y
4	N	N	No	N	N
5	N	Y	Strong	Y	Y
6	N	Y	Strong	N	N
7	Y	Y	Mild	Y	Y
8	N	Y	Strong	Y	Y
9	Y	Y	Mild	Y	Y
10	N	N	No	N	N
11	Y	Y	Strong	Y	Y
12	Y	N	Strong	Y	Y
13	Y	N	Mild	Y	Y

[5]:
```
X = pd.DataFrame()
for x in X_raw:
    X[x] = X_raw[x].map(map_dict)
```

[6]:
```
X.head()
```

[6]:

	Chills	Runny_nose	Headache	Fever
0	1	0	1	1
1	1	1	0	0
2	1	0	2	1
3	0	1	1	1
4	0	0	0	0

[7]:
```
y.head()
```

[7]:
```
0    N
1    Y
2    Y
3    Y
4    N
Name: Flu, dtype: object
```

[8]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.5,
 ↪random_state=42)
model = GaussianNB().fit(X_train, y_train.values.ravel())
predicted_y = model.predict(X_test)
```

[11]:
```
print("Accuracy:", accuracy_score(y_test, predicted_y))
print("Confusion Matrix:\n", confusion_matrix(y_test, predicted_y))
```

```
Accuracy: 0.8571428571428571
Confusion Matrix:
 [[0 1]
 [0 6]]
```

[12]:
```
def make_prediction(case):
    df = pd.DataFrame(case)[0].map(map_dict)
    return model.predict([df])
```

[13]:
```
print(make_prediction(["Y", "N", "Mild", "Y"]))
```

```
['Y']
```

[14]:
```
print(make_prediction(["N", "N", "Mild", "N"]))
```

```
['N']
```

Question 6

k-Nearest Neighbor (k-NN)

6.1 Introduction

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. K-NN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

6.2 Creating Theoretical Model

Step-1: Select the number K of the neighbors. k=5 is preferred.

Step-2: Calculate the Euclidean distance of k number of neighbors.

$$Euclidean\ Distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Step-3: Take the K nearest neighbors as per the calculated Euclidean distance. We may use other means of finding sum of distance calculations as well with Minkowski by setting p to 1 and 2 for using Manhattan and Euclidean, respectivly.

$$||x_1 - x_2|| = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}$$

Step-4: Among these k neighbors, count the number of the data points in each category.

Step-5: Assign the new data points to that category where the neighbor is maximum.

6.3 Implementation

[1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

[2]:
```
dataset = pd.read_csv('./data/ads.csv')
X = dataset.iloc[:, [1, 2]].values
y = dataset.iloc[:, -1].values
dataset.head(10)
```

[2]:
```
   Gender  Age  EstimatedSalary  Purchased
0    Male   19            19000          0
1    Male   35            20000          0
2  Female   26            43000          0
3  Female   27            57000          0
4    Male   19            76000          0
5    Male   27            58000          0
6  Female   27            84000          0
7  Female   32           150000          1
8    Male   25            33000          0
9  Female   35            65000          0
```

[3]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,␣
↪random_state = 0)
```

[4]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[5]:
```
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p =␣
↪2)
classifier.fit(X_train, y_train)
```

[5]:
```
KNeighborsClassifier()
```

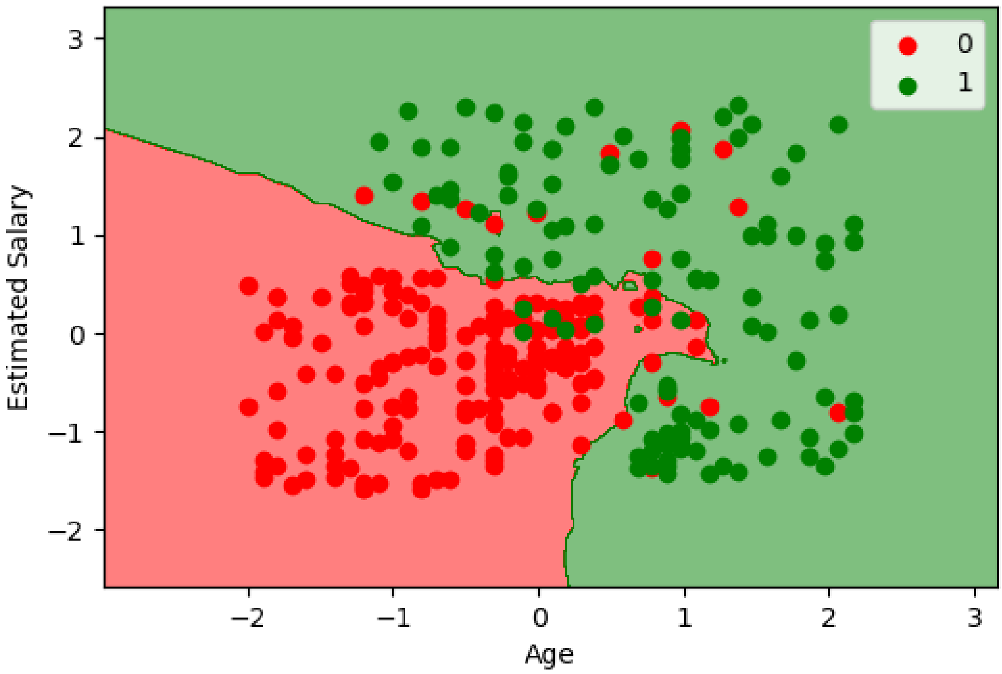
[6]:
```
y_pred = classifier.predict(X_test)
print(y_pred)
print(y_test)
```

```
[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1
 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 1 1]
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1
 0 0 0 0 1 1 1 0 0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 1 1 1]
```

[7]:
```
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test,y_pred))
```

```
[[64  4]
 [ 3 29]]
0.93
```

[8]:
```
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:
 ↪, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:
 ↪, 1].max() + 1, step = 0.01))
plt.figure(dpi=100)
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).
 ↪T).reshape(X1.shape),
             alpha = 0.5, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], color =␣
 ↪ListedColormap(('red', 'green'))(i), label = j)
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



[9]:
```
def make_predection(age, salary):
    return classifier.predict(sc.transform([[age,salary]]))
```

[10]:
```
make_predection(25, 130000)
```

[10]:
```
array([1], dtype=int64)
```

[11]:
```
make_predection(5, 100)
```

[11]:
```
array([0], dtype=int64)
```

Question 7

Fuzzy Control System

7.1 Introduction

A control system is an arrangement of physical components designed to alter another physical system so that this system exhibits certain desired characteristics.

7.2 Steps in Designing Fuzzy Control System

Following are the steps involved in designing Fuzzy Control System -

1. Identification of variables - Here, the input, output and state variables must be identified of the plant which is under consideration.
2. Fuzzy subset configuration - The universe of information is divided into number of fuzzy subsets and each subset is assigned a linguistic label. Always make sure that these fuzzy subsets include all the elements of universe.
3. Obtaining membership function - Now obtain the membership function for each fuzzy subset that we get in the above step. A fuzzy set $\tilde{A}$ in the universe of information U, where $\mu_{\tilde{A}}(y)$ maps U, with respect to y, to membership space.

$$\tilde{A} = \{(y, \mu_{\tilde{A}}(y))|y \in U\}$$

4. Fuzzy rule base configuration - Now formulate the fuzzy rule base by assigning relationship between fuzzy input and output.
5. Fuzzification - The fuzzification process is initiated in this step.
6. Combining fuzzy outputs - By applying fuzzy approximate reasoning, locate the fuzzy output and merge them.
7. Defuzzification - Finally, initiate defuzzification process to form a crisp output.

7.3 Implementation

[1]:
```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

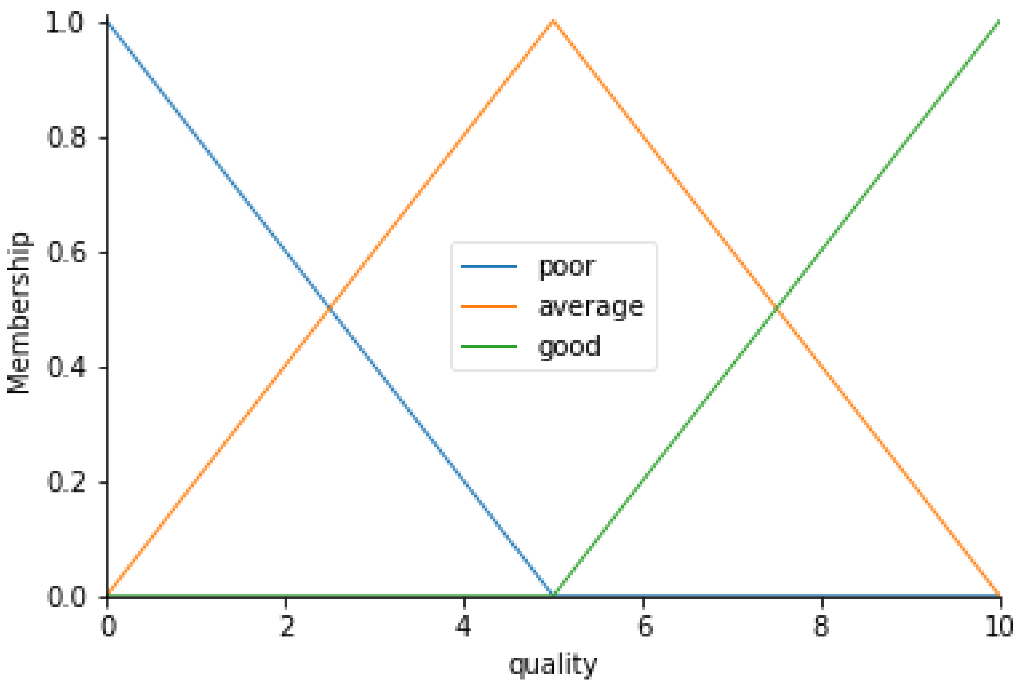
[2]:
```
quality = ctrl.Antecedent(np.arange(0, 11, 1), "quality")
service = ctrl.Antecedent(np.arange(0, 11, 1), "service")
```

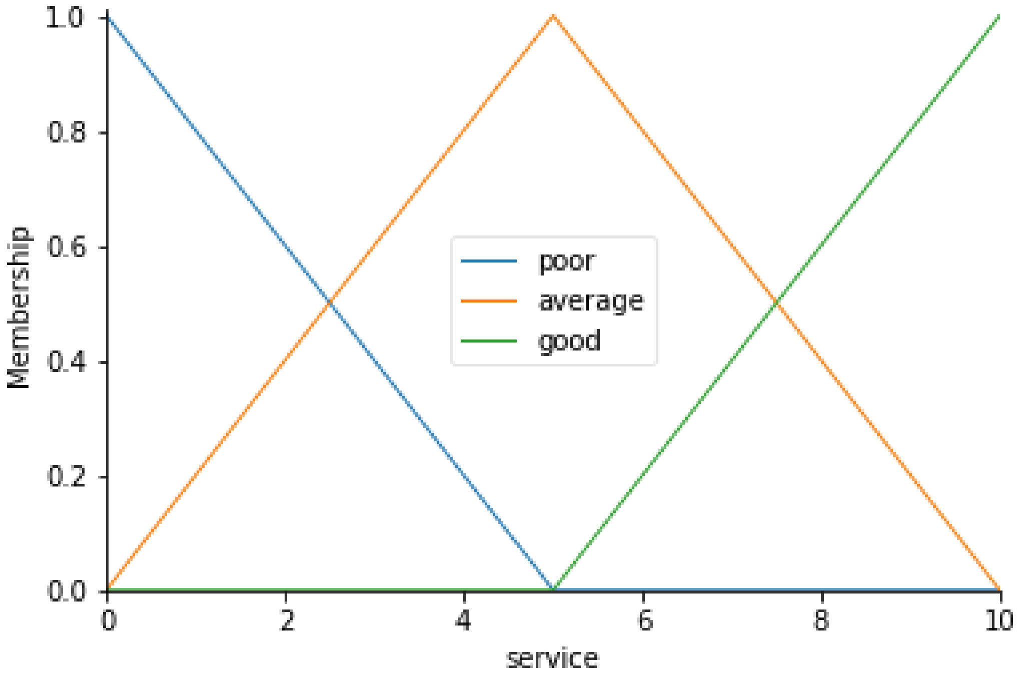
[3]:
```
tip = ctrl.Consequent(np.arange(0, 26, 1), "tip")
```

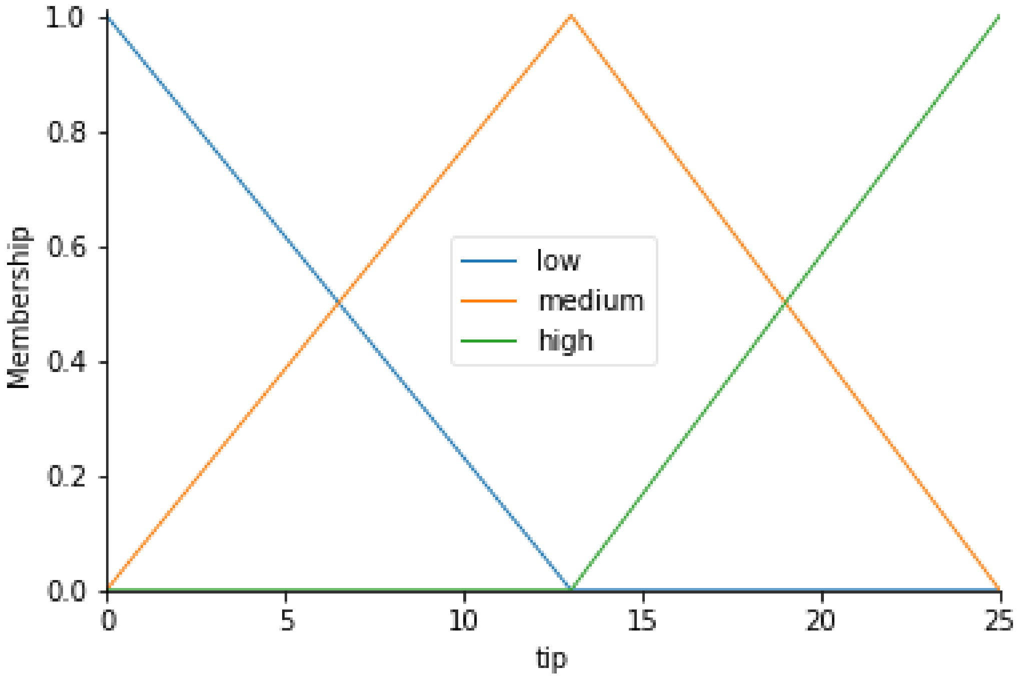
[4]:
```
quality.automf(3) # Poor, average, good
service.automf(3) # Poor, average, good
```

[5]:
```
# Manual Functions
tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])
```

[6]:
```
quality.view()
service.view()
tip.view()
```







[7]:
```
# Inference rule set
rules = [
    ctrl.Rule(quality['poor'] & service['poor'], tip['low']),
    ctrl.Rule(quality['good'] & service['good'], tip['high']),
    ctrl.Rule(quality['average'] | service['good'], tip['medium'])]
```

[8]:
```
tip_ct = ctrl.ControlSystem(rules)
tipping = ctrl.ControlSystemSimulation(tip_ct)
```

[9]:
```
tipping.input['quality'] = 6
tipping.input['service'] = 8
tipping.compute()
print("Tip should be around", tipping.output['tip'])
```

```
Tip should be around 12.889795918367348
```

[10]:
```
tip.view(sim=tipping)
```