Name: Ashish Verma Course:CS773 Exam1

Solution 1

Given

Professional#	Experience (years)	Actual Salary ($K)
1	2	38
2	4	38
3	6	50

Given Linear Equation

salary($K)=30+2.5×Experience (in years)

Professional 1 Predicted Salary

- Predicted Salary=30+2.5×2= **35K**
- MSE=(38−35)^2
- MSE=9

Professional 2 Predicted Salary

- Predicted Salary=30+2.5×4= **30K**
- MSE=(38−40)^2
- MSE=4

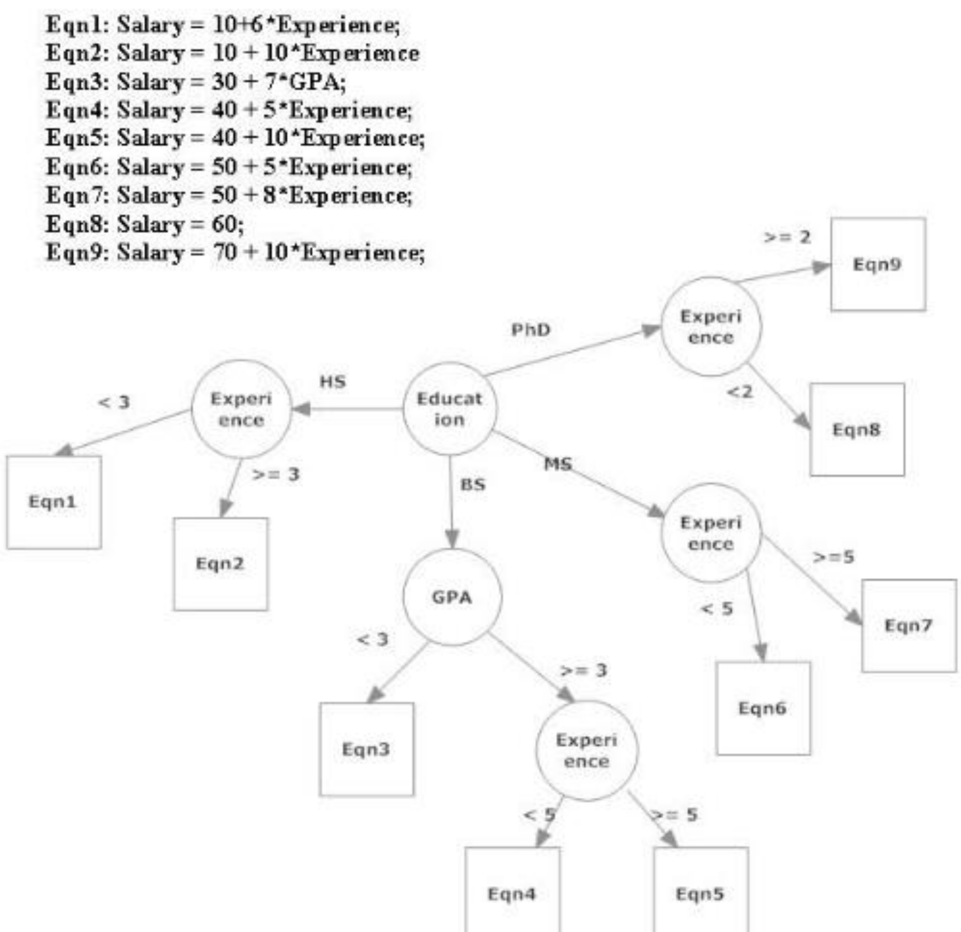
Professional 3 Predicted Salary

- Predicted Salary=30+2.5×6= **45K**
- MSE=(50−45)^2
- MSE=25

Total MSE = 1/3(9+4+25) = **12.67**

Solution 2

Given



Decision Tree Analysis

Applicant (i): Education=MS, GPA=4.0, Experience=8 years

- Education: MS
- GPA: 4.0
- Experience: 8 years

Following the decision tree:

- For MS, GPA >= 3, experience >= 5:
    - Use Eqn7: Salary = 50 + 8 * Experience
    - Salary = 50 + 8 * 8
    - Salary = 50 + 64
    - Salary = 114 K

Applicant (ii): Education=HS, GPA=3.5, Experience=12years

- Education: HS
- GPA: 3.5
- Experience: 12 years

Following the decision tree:

- For HS, experience >= 3:
    - Use Eqn2: Salary = 10 + 10 * Experience
    - Salary = 10 + 10 * 12
    - Salary = 10 + 120
    - Salary = 130 K

Applicant (iii): Education=PhD, GPA=3.0, Experience=0 years

    - Education: PhD
    - GPA: 3.0
    - Experience: 0 years

Following the decision tree:

- For PhD, experience < 2:
    - Use Eqn8: Salary = 60
    - Salary = 60 K

The estimated salaries for the three applicants are:

- Applicant (i): 114 K
- Applicant (ii): 130 K
- Applicant (iii): 60 K

Solution 3

Given

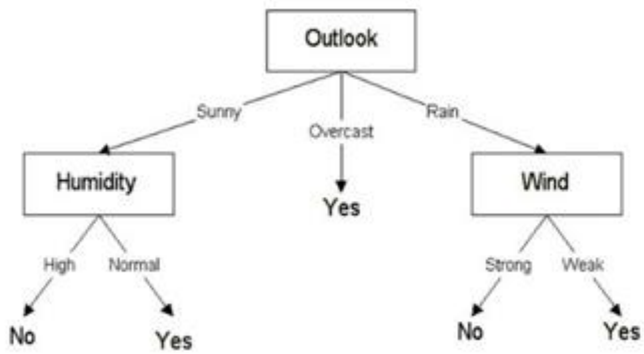
Professional#	Name	Education	Experience (years)
1	John	BS	10
2	Jane	PhD	5
3	Jim	HS	25

```
def predict_salary(education, experience):
    if education >= "BS" and experience < 8:
        salary = 60 + 4 * experience
    elif education == "PhD":
        salary = 80 + 5 * experience
    elif experience > 10:
        if education >= "BS":
            salary = 70 + 7 * experience
        else:
            salary = 50 + 5 * experience
    else:
        salary = 40 + 3 * experience

    return salary

# Predicting salaries for John, Jane, and Jim
john_salary = predict_salary("BS", 10)
jane_salary = predict_salary("PhD", 5)
jim_salary = predict_salary("HS", 25)

# Output the predicted salaries
print(f"Predicted salary for John: ${john_salary}")
print(f"Predicted salary for Jane: ${jane_salary}")
print(f"Predicted salary for Jim: ${jim_salary}")
```

- Predicted salary for John: $70
- Predicted salary for Jane: $80
- Predicted salary for Jim: $245

Solution 4

Given



Derived Rules from the decision tree path

- If the outlook is Sunny and the humidity is High, then Play = No.
- If the outlook is Sunny and the humidity is Normal, then Play = Yes.
- If the outlook is Overcast, then Play = Yes.
- If the outlook is Rain and the wind is Strong, then Play = No.
- If the outlook is Rain and the wind is Weak, then Play = Yes.

Solution 5

Given

Instance#	Price	Capacity	Safety	Acceptability
1	L	4	High	Good
2	H	2	High	Bad
3	M	4	Medium	Good
4	L	7	High	Good
5	M	7	Low	Bad
6	H	7	Medium	Bad

Frequency Table

Acceptability	Price	Capacity	Safety	Count
Good	L	4	High	1
Good	L	7	High	1
Bad	H	2	High	1
Bad	M	7	Low	1
Bad	H	7	Medium	1

Applying Laplace Smoothing

Acceptability	Price	Capacity	Safety	Count
Good	L	4	High	2
Good	L	7	High	2
Bad	H	2	High	2
Bad	M	7	Low	2
Bad	H	7	Medium	2

Calculating Probabilities

For Acceptability = Good:

P(Acceptability=Good|Price=L,Capacity=7,Safety=M)= 0.5×0.67×0.33×0.33=0.0363

For Acceptability = Bad:

P(Acceptability=Bad|Price=L,Capacity=7,Safety=M)= 0.5×0.33×0.67×0.33 = 0.0363

Normalize Probabilities

P(Acceptability=Good|Price=L,Capacity=7,Safety=M) = 0.0363/(0.0363+0.0363) = 0.5

P(Acceptability=Bad|Price=L,Capacity=7,Safety=M) =0.0363/(0.0363+0.0363) = 0.5

Construct the Prediction Table

Instance#	Price	Capacity	Safety	Acceptability (Predicted)	Probability (Good)	Probability (Bad)
1	L	4	High	Good	0.5	0.5
2	H	2	High	Bad	0.5	0.5
3	M	4	Medium	Good	0.5	0.5
4	L	7	High	Good	0.5	0.5
5	M	7	Low	Bad	0.5	0.5
6	H	7	Medium	Bad	0.5	0.5
New	L	7	M	?	0.5	0.5

- Acceptability: **Good**
- Probability (Good): **0.5**
- Probability (Bad): **0.5**

Solution 6

Given

Instance#	Price	Capacity	Safety	Acceptability
1	L	4	High	Good
2	H	2	High	Bad
3	M	4	Medium	Good
4	L	7	High	Good
5	M	7	Low	Bad
6	H	7	Medium	Bad

Since calculations are very long for entropy with given time frame I am going with python based implementation

```python
import pandas as pd
import math

# Define the training data
data = {
    'Price': ['L', 'H', 'M', 'L', 'M', 'H'],
    'Capacity': [4, 2, 4, 7, 7, 7],
    'Safety': ['High', 'High', 'Medium', 'High', 'Low', 'Medium'],
    'Acceptability': ['Good', 'Bad', 'Good', 'Good', 'Bad', 'Bad']
}

# Convert the data into a pandas DataFrame
df = pd.DataFrame(data)

# Calculate the entropy of a target column
def calculate_entropy(target_column):
    entropy = 0
    values = target_column.unique()
    for value in values:
        fraction = target_column.value_counts()[value] / len(target_column)
        entropy += -fraction * math.log2(fraction)
    return entropy

# Calculate information gain for a given attribute
def calculate_information_gain(data, attribute, target):
    # Calculate entropy of the entire dataset
    total_entropy = calculate_entropy(data[target])

    # Calculate weighted entropy for the attribute
    attribute_entropy = 0
    attribute_values = data[attribute].unique()

    for value in attribute_values:
        subset = data[data[attribute] == value]
        fraction = len(subset) / len(data)
        attribute_entropy += fraction * calculate_entropy(subset[target])

    # Calculate information gain
    information_gain = total_entropy - attribute_entropy
    return information_gain

# Calculate information gain for each attribute
attributes = ['Price', 'Capacity', 'Safety']
target = 'Acceptability'

information_gains = {}
for attribute in attributes:
    information_gains[attribute] = calculate_information_gain(df, attribute, target)

print("Information gains:")
for attribute, gain in information_gains.items():
    print(f"{attribute}: {gain}")

# Determine the attribute with the highest information gain (root of the decision tree)
root_attribute = max(information_gains, key=information_gains.get)
print(f"\nRoot of the decision tree: {root_attribute}")
```

```
Information gains:
Price: 0.6666666666666667
Capacity: 0.5408520829727552
Safety: 0.20751874963942196

Root of the decision tree: Price
```

As we can clearly Price is the root

Solution 7

Given

Inst#	Age	Height	Weight	Health
1	35	5.6	175	Excellent
2	55	6	150	Good
3	50	5.8	200	Okay
4	65	5.5	175	Good
5	45	5.6	190	Okay

Normalize the attribute

- Age: 30-70
- Height: 4.0-7.0
- Weight: 100-300

- Normalized Age = 50 -30/70 -30 = 0.5
- Normalized Height = 5.5 - 4.0 /7.0 – 4.0 = 0.5
- Normalized Weight = 190   - 100 /300 – 100 = 0.45

The normalized unknown instance is <0.5,0.5,0.45>

Calculate Euclidean distances

Instance 1

- ED1 = sqrt((0.5 – 0.125) ^2+(0.5 -0.533)^2+(0.45 -0.375)^2)
- ED1 = 0.384

Instance 2

- ED2 = sqrt((0.5 – 0.625) ^2+(0.5 -0.6667)^2+(0.45 -0.25)^2)
- ED2 = 0.289

Instance 3

- ED2 = sqrt((0.5 – 0.5) ^2+(0.5 -0.6)^2+(0.45 -0.5)^2)
- ED2 = 0.10125

Instance 4

- ED2 = sqrt((0.5 – 0.875) ^2+(0.5 -0.5)^2+(0.45 -0.375)^2)
- ED2 = 0.382

Instance 5

- ED2 = sqrt((0.5 – 0.375) ^2+(0.5 -0.5333)^2+(0.45 -0.45)^2)
- ED2 = 0.129

The nearest instance to the unknown instance <0.5,0.5,0.45> is instance 1 with a Euclidean distance of approximately 0.129

Predict Health

Age=50, Height=5.5, Weight=190> is **Okay**.

Solution 8

Given

Instance#	Price	Capacity	Safety	Acceptability
1	L	4	High	Good
2	H	2	High	Bad
3	M	4	Medium	Good
4	L	7	High	Good
5	L	7	Low	Bad
6	H	7	High	Bad

Analyze each attribute

Price:

- L: 3 instances (1, 4, 5)
- M: 1 instance (3)
- H: 2 instances (2, 6)

Capacity:

- 2: 1 instance (2)
- 4: 2 instances (1, 3)
- 7: 3 instances (4, 5, 6)

Safety:

- High: 4 instances (1, 2, 4, 6)
- Medium: 1 instance (3)
- Low: 1 instance (5)

Calculate error rates for each attribute

Price:

- L: Error rate = 1/3 (1 incorrect out of 3) = 0.33

- M: Error rate = 0/1 = 0
- H: Error rate = 2/2 = 1.0

Capacity:

- 2: Error rate = 1/1 = 1.0
- 4: Error rate = 0/2 = 0
- 7: Error rate = 2/3 = 0.67

Safety:

- High: Error rate = 1/4 = 0.25
- Medium: Error rate = 0/1 = 0
- Low: Error rate = 1/1 = 1.0

Select the attribute with the lowest error rate

From the analysis, **Price** has the lowest error rate:

- L: Error rate = 0.33
- M: Error rate = 0
- H: Error rate = 1.0

Therefore, the 1R rule is to predict **Acceptability** based on **Price**:

- **L** predicts **Good**
- **M** predicts **Good**
- **H** predicts **Bad**

Final Selected Rule:

- If **Price = L**, then **Acceptability = Good**
- If **Price = M**, then **Acceptability = Good**
- If **Price = H**, then **Acceptability = Bad**

Solution 9

Given

Inst#	Age	Height	Weight	Predicted Prob.			Actual
				Okay	Good	Excellent	
1	35	5.6	175	0.4	0.4	0.2	Okay
2	55	6	150	0.3	0.2	0.5	Good
3	50	5.8	200	0.1	0.6	0.3	Excellent

Quadratic loss function

Instance 1

- Okay: (0.4−1)^2=(0.6)^2=0.36
- Good: (0.4−0)^2=(0.4)^2=0.16
- Excellent: (0.2−0)^2=(0.2)^2=0.04

MSE = 1/3*(0.26+0.16+0.04) =0.153

Instance 2

- Okay: (0.3−0)^2=(0.3)2=0.09
- Good: (0.2−1)^2=(0.8)2=0.64
- Excellent: (0.5−0)^2=(0.5)2=0.25

MSE = 1/3*(0.09+0.64+0.25) =0.3267

Instance 3

- Okay: (0.1−0)2=(0.1)2=0.01
- Good: (0.6−0)2=(0.6)2=0.36
- Excellent: (0.3−1)2=(0.7)2=0.49

MSE = 1/3*(0.01+0.36+0.49)

=0.2867

Total MSE =(0.153+0.3267+0.2867)/3

=**0.2554**

Information loss function

Instance 1

Loss=−(1*log_2(0.4)+0*log_2(0.4)+0*log_2(0.2))

=1.3219

Instance 2

Loss=−(0*log_2(0.3)+1*log_2(0.2)+0*log_2(0.5))

=2.3219

Instance 3

Loss = −(0*log_2(0.1)+0*log_2(0.6)+1*log_2(0.3))

=1.7369

Total Entropy loss =(1.3219+2.3219 +1.7369)/3

=**1.7936**

Solution 10

Given

- Total number of volunteers (N) = 2500
- Number of volunteers who actually have the disease (Disease positive) = 1500
- Number of volunteers identified by the test as having the disease (Test positive) = 1750
- Number of volunteers correctly identified as having the disease (True positives, TP) = 1000

- TP = 1000
- FN = 1500 – 1000 = 500
- TN = 2500−1500−(1750−1000)=1000
- FP= 1750−1000=750

Sensitivity = TP/(TP+FN) = 1000/1000+500 = 0.667

Specificity = TN/(TN+FP) = 1000/1000+750 = 0.571

Solution 11

Given

Customer#	Predicted Prob (Yes)	Actual
1	0.85	Yes
2	0.5	No
3	0.95	No
4	0.99	Yes
5	0.45	Yes
6	0.97	No
7	0.8	Yes
8	0.6	No
9	0.75	Yes
10	0.7	No

Python

```python
import numpy as np
import pandas as pd
from sklearn.metrics import roc_curve

# Customer data
customers = [
    {"predicted_prob": 0.85, "actual": "Yes"},
    {"predicted_prob": 0.5, "actual": "No"},
    {"predicted_prob": 0.95, "actual": "No"},
    {"predicted_prob": 0.99, "actual": "Yes"},
    {"predicted_prob": 0.45, "actual": "Yes"},
    {"predicted_prob": 0.97, "actual": "No"},
    {"predicted_prob": 0.8, "actual": "Yes"},
    {"predicted_prob": 0.6, "actual": "No"},
    {"predicted_prob": 0.75, "actual": "Yes"},
    {"predicted_prob": 0.7, "actual": "No"}
]

# Sort customers by predicted probability (descending)
customers.sort(key=lambda x: x["predicted_prob"], reverse=True)

# Extract actual labels and predicted probabilities
actual = np.array([1 if c["actual"] == "Yes" else 0 for c in customers])
predicted_prob = np.array([c["predicted_prob"] for c in customers])

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(actual, predicted_prob)

# Create a DataFrame for the ROC curve table
roc_df = pd.DataFrame({
    'Threshold': np.concatenate(([1.0], thresholds)),
    'TPR': np.concatenate(([0.0], tpr)),
    'FPR': np.concatenate(([0.0], fpr))
})

# Print the ROC curve table
print(roc_df)
```

	Threshold	TPR	FPR
0	1.00	0.0	0.0
1	1.99	0.0	0.0
2	0.99	0.2	0.0
3	0.95	0.2	0.4
4	0.75	0.8	0.4
5	0.50	0.8	1.0
6	0.45	1.0	1.0

Sort the data with predicted probability in descending order

Customer#	Predicted Prob (Yes)	Actual
4	0.99	Yes
6	0.97	No
3	0.95	No
7	0.8	Yes
1	0.85	Yes
9	0.75	Yes
10	0.7	No
8	0.6	No
2	0.5	No
5	0.45	Yes

Now, calculate TPR, FPR, and the threshold for each data point:

Threshold = 1.0 (All predicted as Yes)

- TPR = 4/4 = 1.00
- FPR = 6/6 = 1.00

Threshold = 0.99

- TPR = 4/4 = 1.00
- FPR = 6/6 = 1.00

Threshold = 0.95

- TPR = 4/4 = 1.00
- FPR = 5/6 = 0.833

Threshold = 0.85

- TPR = 4/4 = 1.00

- FPR = 3/6 = 0.500

Threshold = 0.8

- TPR = 4/4 = 1.00
- FPR = 2/6 = 0.333

Threshold = 0.75

- TPR = 4/4 = 1.00
- FPR = 1/6 = 0.167

Threshold = 0.7

- TPR = 3/4 = 0.75
- FPR = 1/6 = 0.167

Threshold = 0.6

- TPR = 3/4 = 0.75
- FPR = 1/6 = 0.167

Threshold = 0.5

- TPR = 3/4 = 0.75
- FPR = 1/6 = 0.167

Threshold = 0.45

- TPR = 3/4 = 0.75
- FPR = 1/5 = 0.200

Threshold = 0.0 (All predicted as No)

- TPR = 0/4 = 0.00
- FPR = 0/6 = 0.00

Threshold	TPR	FPR
1	1	1
0.99	1	1
0.95	1	0.833
0.85	1	0.5
0.8	1	0.333
0.75	1	0.167
0.7	0.75	0.167
0.6	0.75	0.167
0.5	0.75	0.167
0.45	0.75	0.2
0	0	0

Solution 12

Given

Instance#	Actual	Predicted
1	2.5	3
2	4	4.3
3	3.5	2.5
4	5	3

MAE = (2.5-3 + 4.3-4+ 3.5-2.5 + 5-3)/4 = 0.95

So, the mean absolute error for your predictions is **0.95**

RAE = (0.2+0.075+0.286+0.4)/4 = 0.25 = **25%**

Solution 13

Given

instance#	Age	GPA	Salary($K)	Cluster#
1	25	3.8	50	C1
2	30	3.6	65	C1
3	23	4	40	C1
4	35	3	70	C2
5	55	3.3	90	C2

Normalized Data

Instance	Normalized Age	Normalized GPA	Normalized Salary	Cluster
1	0.1	0.95	0.375	C1
2	0.2	0.9	0.5625	C1
3	0.06	1	0.25	C1
4	0.3	0.75	0.625	C2
5	0.7	0.825	0.875	C2

Single-linkage method:

Distance(C1_1, C2_1):

Distance(C11,C21) = sqrt((0.1 – 0.3)^2 + (0.95-0.75)^2 +(0.375 -0.625)^2)

=0.574

Distance(C1_1, C2_2):

Distance(C11,C22) = sqrt((0.1 – 0.7)^2 + (0.95-0.825)^2 +(0.375 -0.875)^2)

= 0.796

Distance(C1_2, C2_1)

Distance(C12,C21) = sqrt((0.2 – 0.3)^2 + (0.9-0.75)^2 +(0.5625 -0.625)^2)

= 0.191

Distance(C1_2, C2_2)

Distance(C13,C21) = sqrt((0.06 – 0.3)^2 + (1.0-0.75)^2 +(0.25 -0.625)^2)

= 0.55

Distance(C1_3, C2_2):

Distance(C13,C22) = sqrt((0.06 – 0.7)^2 + (1.0-0.825)^2 +(0.25 -0.825)^2)

=0.92

Minimum distance = min(0.574, 0.796, 0.191, 0.785, 0.55, 0.92)

Minimum distance ≈ 0.191

Centroid-linkage method:

Centroid-linkage method computes the distance between the centroids of the two clusters.

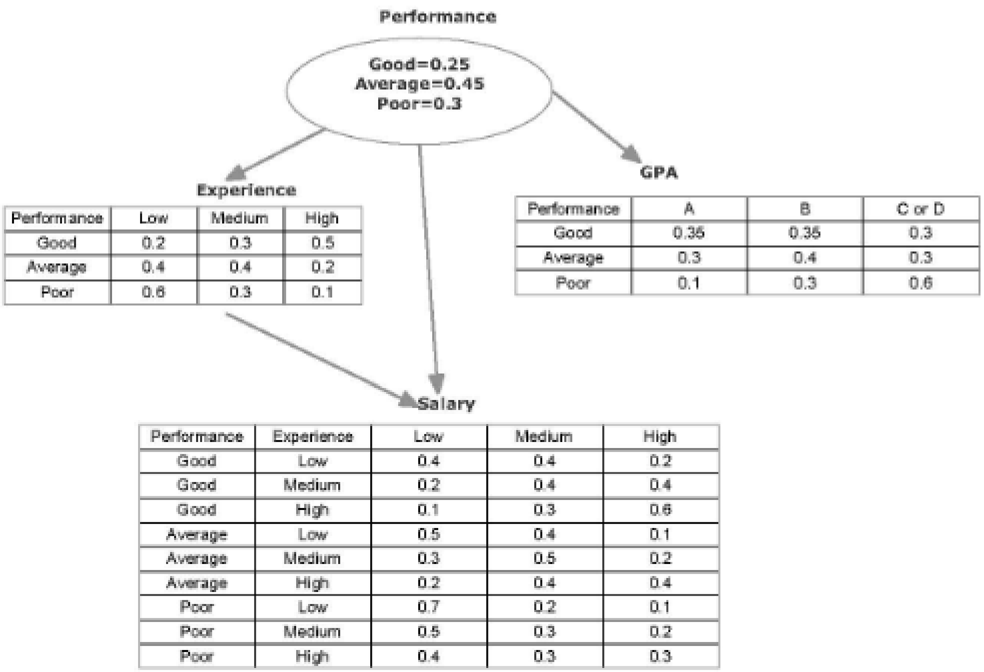
Distance using Centroid-linkage method:

- Distance = sqrt((0.12 - 0.5)^2 + (0.95 - 0.7875)^2 + (0.394 - 0.75)^2)
- Distance = sqrt(0.1696 + 0.02600625 + 0.129936)
- Distance = sqrt(0.32553625)
- Distance ≈ 0.571

Therefore, the distance between the centroids of clusters C1 and C2 using the centroid-linkage method is approximately **0.571**.

Solution 14

Given



Performance	Low	Medium	High
Good	0.2	0.3	0.5
Average	0.4	0.4	0.2
Poor	0.6	0.3	0.1

Performance	A	B	C or D
Good	0.35	0.35	0.3
Average	0.3	0.4	0.3
Poor	0.1	0.3	0.6

Performance	Experience	Low	Medium	High
Good	Low	0.4	0.4	0.2
Good	Medium	0.2	0.4	0.4
Good	High	0.1	0.3	0.6
Average	Low	0.5	0.4	0.1
Average	Medium	0.3	0.5	0.2
Average	High	0.2	0.4	0.4
Poor	Low	0.7	0.2	0.1
Poor	Medium	0.5	0.3	0.2
Poor	High	0.4	0.3	0.3

For Performance = Good:

$$P(Good,GPA=B,Experience=High,Salary=Low)=0.35\times0.5\times0.1\times0.25=0.004375$$

For Performance = Average:

$$P(Average,GPA=B,Experience=High,Salary=Low)=0.4\times0.2\times0.2\times0.45=0.0072$$

For Performance = Poor:

$$P(Poor,GPA=B,Experience=High,Salary=Low)=0.3\times0.1\times0.4\times0.3=0.0036$$

Normalization

$$P(GPA=B,Experience=High,Salary=Low)=0.004375+0.0072+0.0036=0.015175$$

Conditional probabilities:

- P(Good|GPA=B,Experience=High,Salary=Low) = 0.004375/ 0.015175 = 0.288
- P(Average|GPA=B,Experience=High,Salary=Low) = 0.0072/0.015175 = 0.474
- P(Poor|GPA=B,Experience=High,Salary=Low) = 0.0036/0.015175 = 0.237

Good: 0.288
Average: 0.474
Poor: 0.237

Solution 15

Given

Instance#	Color	Size	Act	Age	Inflated (outcome)
1	Y	S	S	A	Yes
2	Y	S	S	C	Yes
3	Y	S	D	A	Yes
4	Y	S	S	C	No
5	Y	L	S	A	Yes
6	P	L	S	C	Yes
7	P	L	D	A	No
8	P	L	D	C	No
9	P	S	S	A	Yes
10	P	S	S	C	No

From the dataset, the instances that satisfy the rule "Color=Y and Size=S and Act=S and Age=A and Inflated=Yes" are:

- Instance 1
- Instance 2
- Instance 5
- Instance 9

So, the number of instances that satisfy the rule = 4

Total number of instances in the dataset = 10

Support = 4 / 10 = 0.4

Accuracy:

From the instances that satisfy the rule:

- Instance 1: Inflated=Yes
- Instance 2: Inflated=Yes
- Instance 5: Inflated=Yes
- Instance 9: Inflated=Yes

Number of instances that satisfy the rule and have Inflated=Yes = 4

Accuracy = 4 / 4 = 1.0

Therefore, the support of the rule is **0.4** and the accuracy of the rule is **1.0**.

Given conditions:

- Color = Y
- Size = S
- Act = S
- Age = A
- Inflated = Yes

Instances that match these conditions:

- Instance 1: Color=Y, Size=S, Act=S, Age=A, Inflated=Yes
- Instance 2: Color=Y, Size=S, Act=S, Age=C, Inflated=Yes
- Instance 5: Color=Y, Size=L, Act=S, Age=A, Inflated=Yes

Now, calculate the support and accuracy:

Support: Number of instances that match the rule = 3 (instances 1, 2, and 5)

Accuracy: Percentage of instances that are Inflated = Yes among those that match the rule

Out of the 3 instances that match the rule, all 3 have Inflated = Yes.

Accuracy = (3 / 3) * 100 = 100%

Since the accuracy is 100%, we don't need to prune based on accuracy. Let's check the support:

Support = 3

The rule already meets the minimum support (3) and accuracy (100%) criteria. Therefore, the rule is:

Color = Y AND Size = S AND Act = S AND Age = A => Inflated = Yes

Solution 16

Given

#	COLOR	AGE	RESULT
1	YELLOW	ADULT	F
2	YELLOW	CHILD	T
3	YELLOW	ADULT	T
4	RED	ADULT	F
5	RED	CHILD	F
6	RED	ADULT	T
7	RED	CHILD	F
8	RED	ADULT	F
9	RED	CHILD	T
10	RED	ADULT	F

Frequency Count

Item	Frequency
YELLOW	3
RED	7
ADULT	6
CHILD	4
F	6
T	4

Filtering Items

Items that meet the minimum threshold of 5 are:

- RED
- ADULT
- F

Filtered Transaction

Based on the filtered items, the relevant transactions are:

#	COLOR	AGE	RESULT
4	RED	ADULT	F
6	RED	ADULT	T
8	RED	ADULT	F
10	RED	ADULT	F

FP-tree Construction

- **Insert {RED, ADULT, F}**
- **Insert {RED, ADULT}**
- **Insert {RED, ADULT, F}**
- **Insert {RED, ADULT, F}**

FP-TREE

null

|

RED (4)

|

ADULT (4)

|

F (3)

Solution 17

Given

Attr	Class
80	T
50	F
65	F
40	F
25	T
119	T

Split Point: Between 25 and 40

Partitions:

- Left: <25, T>
- Right: <40, F>, <50, F>, <65, F>, <80, T>, <110, T>

H(Left) = 0

Right partition:

- T: 2 (80, 110)
- F: 3 (40, 50, 65)

H(Right) = −(0.4*log_20.4+0.6*log_20.6)

H(Right) = 0.971

H(Split) =1/6*0+5/6*0.971

=**0.809**

Split Point: Between 40 and 50

Partitions:

- Left: <25, T>, <40, F>
- Right: <50, F>, <65, F>, <80, T>, <110, T>

H(Left) = -(0.5*log_2(0.5)+0.5*log_2(0.5))

=1

Right partition:

- T: 2 (80, 110)
- F: 2 (50, 65)

H(Right) = -(0.5*log_2(0.5)+0.5*log_2(0.5))

=1

H(Split) = 2/6*1 + 4/6*1

=1

Split Point: Between 50 and 65

Partitions:

- Left: <25, T>, <40, F>, <50, F>
- Right: <65, F>, <80, T>, <110, T>

H(Left) =−(0.333*log_2(0.333)+0.667*log_2(0.667))

= 0.918

H(Right) = −(0.667*log_2(0.667)+0.333*log_2(0.333))

=0.918

H(Split) = 3/6*0.918 + 3/6*0.918

=0.918

Split Point: Between 65 and 80

Partitions:

- Left: <25, T>, <40, F>, <50, F>, <65, F>
- Right: <80, T>, <110, T>

H(Left)=−(0.25*log_2(0.25)+0.75*log_2(0.75))

H(Left)=0.811

H(Right)=0

H(Split) = 4/6*0.811+0

H(Split) = 0.541

Split Point: Between 80 and 110

Partitions:

- Left: <25, T>, <40, F>, <50, F>, <65, F>, <80, T>
- Right: <110, T>

H(Left)=−(0.4*log_2(0.4)+0.6*log_2(0.6))

H(Left)=0.971

H(Right)=0 (since it has only one class)

H(Split) = 5/6*0.971 + 0

=0.809

Hence minimum entropy we got between 65 and 80 as 0.541

First point of division =(65+80)/2 = 72.5

Python Implementation

```python
import math

# Given data
data = [
    (80, 'T'),
    (50, 'F'),
    (65, 'F'),
    (40, 'F'),
    (25, 'T'),
    (119, 'T')
]

# Sort data by attribute value
data_sorted = sorted(data, key=lambda x: x[0])

# Calculate entropy function
def entropy(prob):
    if prob == 0 or prob == 1:
        return 0
    return -prob * math.log2(prob) - (1 - prob) * math.log2(1 - prob)

# Function to calculate weighted entropy
def weighted_entropy(groups):
    total_instances = sum(len(group) for group in groups)
    weighted_entropy_sum = 0

    for group in groups:
        if len(group) == 0:
            continue
        count_true = sum(1 for instance in group if instance == 'T')
        prob_true = count_true / len(group)
        group_entropy = entropy(prob_true)
        weighted_entropy_sum += (len(group) / total_instances) * group_entropy

    return weighted_entropy_sum

# Find the optimal split point
min_weighted_entropy = float('inf')
best_split_point = None

# Iterate through each possible split point
for i in range(len(data_sorted) - 1):
    split_point = (data_sorted[i][0] + data_sorted[i + 1][0]) / 2

    # Split the data
    group1 = [item[1] for item in data_sorted[:i+1]]
    group2 = [item[1] for item in data_sorted[i+1:]]

    # Calculate weighted entropy
    we = weighted_entropy([group1, group2])

    # Update minimum weighted entropy and best split point
    if we < min_weighted_entropy:
        min_weighted_entropy = we
        best_split_point = split_point

# Print results
best_split_point, min_weighted_entropy
```

```
(72.5, 0.5408520829727552)
```

Hence best split point is 72.5 and minimum entropy is 0.54075

Solution 18

Given

Class	Class Vector
Good	11100111
Average	00011000
Poor	10101010

Hamming Distance Calculation

Good Vs Average:

- Good: 11100111
- Average: 00011000

Hamming distance = 5 (positions with differences)

Good vs. Poor:

- Good: 11100111
- Poor: 10101010

Hamming distance = 4

Average vs. Poor:

- Average: 00011000
- Poor: 10101010

Hamming distance = 6

Minimum Hamming distance (d) = 4

Error Calculations=(4-1)/2 = 1.5=1

- Hamming Distance (d): 4
- Number of errors that can be corrected: 1

Solution 19

Given

#	Attr Value	Outcome
1	0	T
2	13	F
3	15	T
4	22	F
5	40	F
6	20	T
7	17	F
8	18	F
9	60	T

Equal-frequency binning

Binning:

- **C1**: [0, 13, 15]
- **C2**: [17, 18, 20]
- **C3**: [22, 40, 60]

Attribute Value Range and Sample Outcome for Each Category:

- **C1**: 0-15, Sample Outcome: T (from 0)
- **C2**: 17-20, Sample Outcome: F (from 17)
- **C3**: 22-60, Sample Outcome: F (from 22

Equal-width binning

Width = (Max Value - Min Value) / Number of Bins

= (60 - 0) / 3

= 20

Binning:

- **C1**: [0, 20)
- **C2**: [20, 40)
- **C3**: [40, 60]

Attribute Value Range and Sample Outcome for Each Category:

- **C1**: 0-19, Sample Outcome: T (from 0)
- **C2**: 20-39, Sample Outcome: T (from 20)
- **C3**: 40-60, Sample Outcome: F (from 40)

Solution 20

An approach to transforming the attributes Color (Yellow, Red, Purple) and Size (SM, MD, LG) into a single attribute is to use a hierarchical encoding method. This involves creating a composite key using both attributes while preserving their original structures, which can then be used as a single unique identifier.

Approach: Composite Key Encoding

1. **Concatenate Color and Size**: Combine the two attributes into a single string with a separator.

Combined Attribute

- Yellow_SM
- Red_MD
- Purple_LG

Justification

1. **Preservation of Information**: This method retains the distinct identity of each combination of Color and Size without losing any information.
2. **Ease of Decoding**: The combined attribute can be easily split back into the original attributes if needed, preserving the flexibility of the dataset.
3. **Efficiency in Modeling**: This combined attribute can be treated as a categorical variable in machine learning models, facilitating better handling of the dataset.

Example

Original Data:

- Color: Yellow, Size: SM
- Color: Red, Size: MD
- Color: Purple, Size: LG

Transformed Data:

- Combined Attribute: Yellow_SM
- Combined Attribute: Red_MD
- Combined Attribute: Purple_LG

This approach maintains a clear and distinct identification of each combination of attributes while allowing for efficient data processing and analysis.