classmate
Date
Page

21/9/21 DWDM

Tutorial-5

U18CO021 - Sahil Bondre

Q1 Converting decision tree to rules is preferred because:

a Converting to rules allows distinguishing among different contexts in which a decision node is used

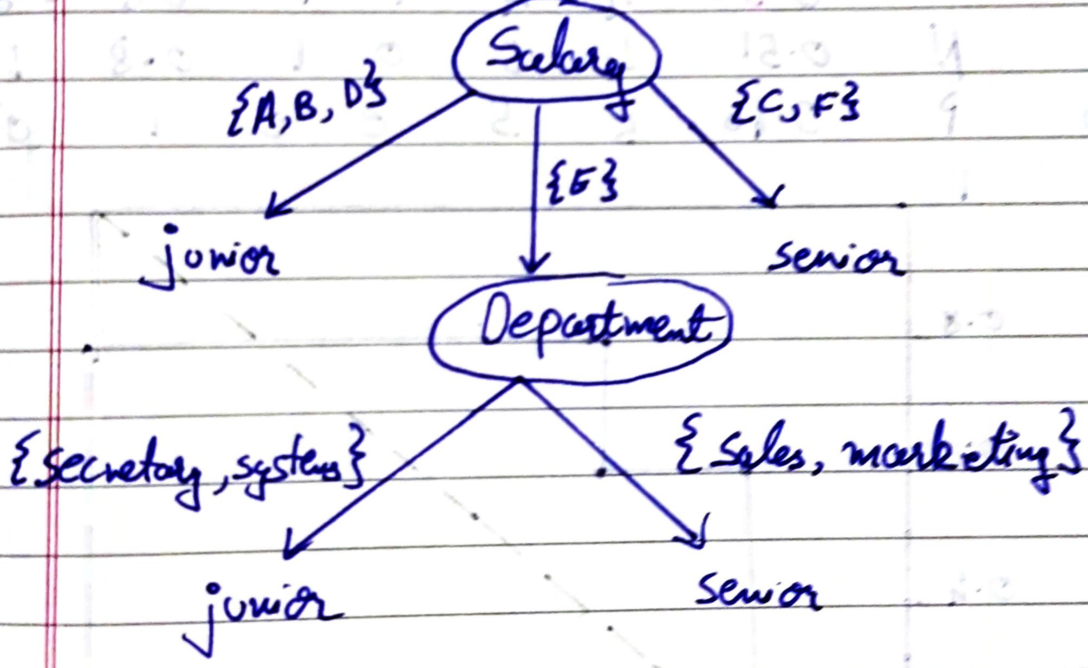
b It removes the distinction between attribute tests occurring at the ~~roo~~ root & leaves of the tree

c It improves readability

Q2

a. To consider count of data tuple, it needs to be ~~such~~ included in the attribute selection measure (such as information gain)

b. Salary Ranges: A = {26-30k}, B = {31-35k}
C = {36-40k}, D = {41-45k}
E = {46-50k}, F = {66-70k}



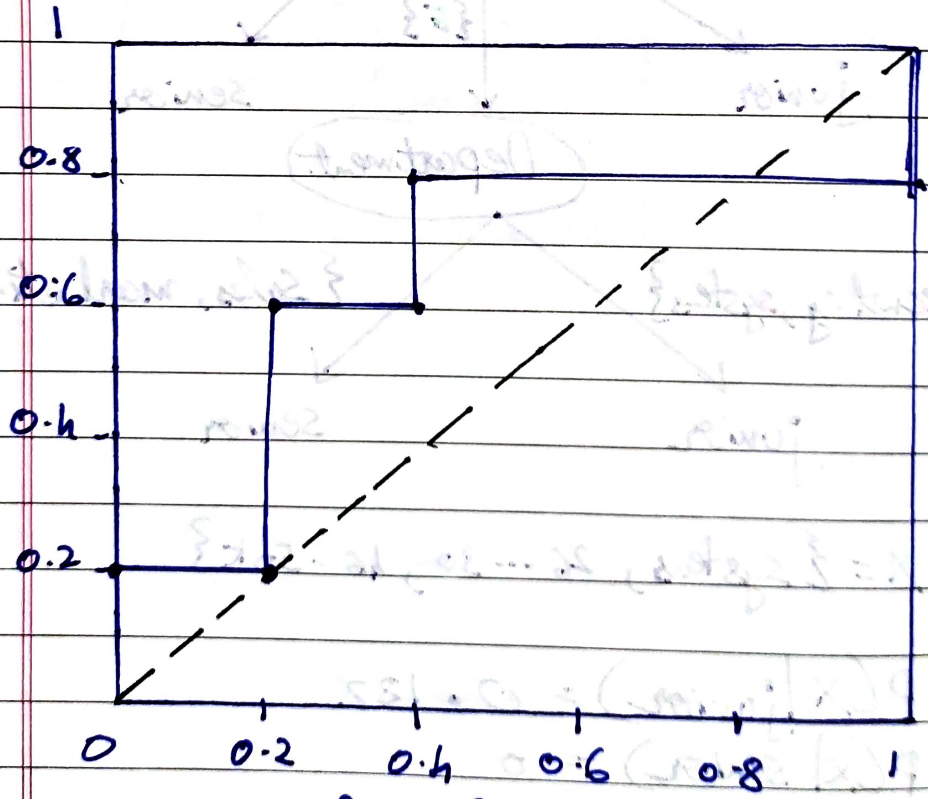
c. X = {systems, 26...30, 46-50k}

P(X|junior) = 0.182
P(X|senior) = 0

∴ NB predicts "junior"

Q3

Tuple	Class	P	TP	FP	TN	FN	TPR	FPR
1	P	0.95	1	0	5	4	0.2	0
2	N	0.85	1	1	4	4	0.2	0.2
3	P	0.78	2	1	4	3	0.4	0.2
4	P	0.76	3	1	4	2	0.6	0.2
5	N	0.60	3	2	3	2	0.6	0.4
6	P	0.55	4	2	3	1	0.8	0.4
7	N	0.53	4	3	2	1	0.8	0.6
8	N	0.52	4	4	1	1	0.8	0.8
9	N	0.51	4	5	0	1	0.8	1
10	P	0.40	5	5	0	0	1	0



ROC Curve

Q4 Methods for class imbalance :

i Oversampling : Resample random tuples from minority class

ii Undersampling : Delete random tuples from majority class.

iii Threshold Moving : Shifting the threshold to reduce the costly false negative errors.

iv Cost-Modifying : Incresing the cost of mis-classifying the minority class

v Ensemble methods : Using boosting and random forests.

SVM & MLP would perform better on imbalanced datasets than tree based classifiers. SVM is not sensitive to imbalance as support vectors only look for boundries between clusters.

Oversampling perfors better for numeric datasets like ~~ex~~ credit card fraud transaction. Cost Modifying would help too.

DWDM Tutorial 5

U18CO021: SAHIL BONDRE

5. Experiment with any dataset of your choice using a decision tree model, logistic regression, naïve Bayes classifier, and a support vector model. Make a table of your results. Are the differences in model performance significant?

1 Import Libraries

[1]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

2 Load Data

[2]:
```
# IRIS Dataset
X, y = load_iris(return_X_y=True)
print(X.shape, y.shape)
```

```
(150, 4) (150,)
```

3 Pre Processing

[3]:
```
# No NAN Values
np.isnan(X).all() or np.isnan(y).all()
```

[3]:
```
False
```

[4]:
```
# Splitting Test Train Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

[5]:
```
# Normalisation
scalar_X = StandardScaler()

X_train = scalar_X.fit_transform(X_train)
X_test = scalar_X.transform(X_test)
```

4 Training

[6]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score
```

[7]:
```python
dt = DecisionTreeClassifier()
lr = LogisticRegression()
nb = GaussianNB()
svm = SVC()
```

[8]:
```python
# Training
dt.fit(X_train, y_train)
lr.fit(X_train, y_train)
nb.fit(X_train, y_train)
svm.fit(X_train, y_train)
```

[8]:
```
SVC()
```

5 Analysis

[9]:
```python
y_pred_dt = dt.predict(X_test)
y_pred_lr = lr.predict(X_test)
y_pred_nb = nb.predict(X_test)
y_pred_svm = svm.predict(X_test)
```

[10]:
```python
pred = [y_pred_dt, y_pred_lr, y_pred_nb, y_pred_svm]

f1 = [f1_score(y_test, y, average="macro") for y in pred]
acc = [accuracy_score(y_test, y) for y in pred]
```

[11]:
```python
print("\033[1m F1 Score \033[0m")
classifier = ["Decesion Tree", "Logistic Regression", "Naive Bayes", "SVM"]

for i in range(len(f1)):
    print(f"{classifier[i]}: {f1[i]:0.2f}")

print()
print("\033[1m Accuracy \033[0m")

for i in range(len(f1)):
    print(f"{classifier[i]}: {acc[i]:0.2f}")
```

```
 F1 Score
Decesion Tree: 0.97
```

```
Logistic Regression: 0.97
Naive Bayes: 0.97
SVM: 0.94

 Accuracy
Decesion Tree: 0.97
Logistic Regression: 0.97
Naive Bayes: 0.97
SVM: 0.93
```