Car Evaluation Example For Non-numerical Data

There are 5 features in the dataset:

buying - buying price
maint - price of the maintenance
doors - number of doors
persons - capacity in terms of persons to carry
lug_boot - the size of luggage boot
safety - estimated safety of the car

All these 5 features are categorical data. The target value is the evaluation level of the car.

Part 1. Decision Tree for Non-numerical Data

Given scikit-learn implementation of decision tree/random forest does not support non-numerical variables, we provide an implementation of decision tree in the folder/package 'decisiontreec'. The idea is from https://github.com/riccardobucco/CDT (https://github.com/riccardobucco/CDT), where the author implemented the algorithm of random forest for categorical data. Here we modified the functions and let it work as a decision tree algorithm for non-numerical data, please note that the split condition is based on the **largest information gain** for the categorical targets.

In [2]:
```
import pandas as pd
from decisiontreec.classifier import _decision_tree, _decision_tree_classify, get_acc
uracy
from decisiontreec.utilities import get_dataset, export_graphviz
```

In [3]:
```
df = pd.read_csv("car.csv")
df.head(5)
```

Out[3]:

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

In [4]:
```
# Split the data into 75% training and 25% test
train, test = get_dataset("car.csv","class",0.75)
```

In [5]:
```
# Dataset - Object : we can get the instance by index using get_instance
# Data instance - Object: can get accessed to the feature values by field 'attribute
s'
train.get_instance(0).attributes
```

Out[5]:
```
{'buying': 'vhigh',
 'maint': 'vhigh',
 'doors': '2',
 'persons': '2',
 'lug_boot': 'small',
 'safety': 'med'}
```

In [6]:
```python
# Construct the decision tree by training data
clf = _decision_tree(train)

# Get instances (each row) from the test set
instances = test._get_instances()
pred_list = []
for instance in instances:
    # Make prediction on each instance in the test set
    pred = _decision_tree_classify(clf,instance)
    pred_list.append(pred)
```

In [8]:
```python
# Calculate the accuracy
print("the accuracy of the classification:", get_accuracy(test,pred_list))
# Visualise the decision tree
# End Node ID [target value]
# Decision Node ID  -> Child Node ID [split condition]
# Try text description of the tree by:
# export_graphviz(clf)
```

```
the accuracy of the classification: 0.8564814814814815
```

Please note that this is not a perfect implementation of decision tree, the problems are: (1) It can only deal with non-numerical data, (2) When the the feature set A is empty, it cannot mark the leaf node to the majority class in D (please refer to an example in week10 lecture slides: lec9-classification3, p.5).

If you are interested in solving the problems mentioned above, please have a look at the functions **_decision_tree** and **_information_gain** in classifier.py, and try to modify the algorithms to make the model more adaptable.

Part 2. Naive Bayes for Non-numerical Data

The categorical Naive Bayes classifier is suitable for classification with **discrete features** that are categorically distributed. So for non-numerical data, we can use LabelEncoder to encode the categorical data into dicrete integers, then fit the data with CategoricalNB.

In [17]:
```python
from sklearn.naive_bayes import CategoricalNB
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [10]:
```python
df = pd.read_csv("car.csv")
df.head(5)
```

Out[10]:

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

In [11]:
```
# Transform categorical data
df = df.apply(LabelEncoder().fit_transform)
df
```

Out[11]:

	buying	maint	doors	persons	lug_boot	safety	class
0	3	3	0	0	2	2	2
1	3	3	0	0	2	0	2
2	3	3	0	0	1	1	2
3	3	3	0	0	1	2	2
4	3	3	0	0	1	0	2
...	...	...	...	...	...	...	...
1722	1	1	3	2	1	2	1
1723	1	1	3	2	1	0	3
1724	1	1	3	2	0	1	2
1725	1	1	3	2	0	2	1
1726	1	1	3	2	0	0	3

1727 rows × 7 columns

In [12]:
```
# Split the dataset into X and y for classification
# Select the last column as label
y = df['class'].values
# Select column 0~5 as features
X = df.iloc[:,0:6].values
```

In [15]:
```
# Split the dataset into train and test set (default train/test is set as 75%/25%)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
print("The number of training data: ", X_train.shape[0], "\nThe number of test data:
 ",X_test.shape[0])
```

```
The number of training data:  1295
The number of test data:  432
```

In [18]:
```
# Construct the model and fit with the training data
cnb = CategoricalNB()
y_pred = cnb.fit(X_train, y_train).predict(X_test)

# summarize the fit of the model
print(metrics.classification_report(y_test, y_pred))
print(metrics.confusion_matrix(y_test, y_pred))

acc = metrics.classification_report(y_test, y_pred, output_dict=True)['accuracy']
print("The prediction accuracy is: ", acc)

print("Number of mislabeled points out of a total %d points : %d" % (X_test.shape[0],
(y_test != y_pred).sum()))
```

```
              precision    recall  f1-score   support

           0       0.61      0.70      0.65        96
           1       0.57      0.22      0.32        18
           2       0.92      0.94      0.93       298
           3       0.88      0.35      0.50        20

    accuracy                           0.83       432
   macro avg       0.74      0.55      0.60       432
weighted avg       0.83      0.83      0.82       432

[[ 67   3  26   0]
 [ 13   4   0   1]
 [ 17   0 281   0]
 [ 13   0   0   7]]
The prediction accuracy is:  0.8310185185185185
Number of mislabeled points out of a total 432 points : 73
```

How to deal with mix data (continous + non-numerical) in Naive Bayes

Naive Bayes based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features - meaning you calculate the Bayes probability dependent on a specific feature without holding the others - which means that the algorithm multiply each probability from one feature with the probability from the second feature (and we totally ignore the denominator - since it is just a normalizer).

so the right answer is:

1. calculate the probability from the non-numerical variables.
2. calculate the probability from the continuous variables.
3. multiply 1. and 2.