Section 5: Machine Learning

Using the dataset from https://archive.ics.uci.edu/ml/datasets/Car+Evaluation, create a machine learning model to predict the buying price given the following parameters:

- Maintenance = High
- Number of doors = 4
- Lug Boot Size = Big
- Safety = High
- Class Value = Good

Note: This notebook was ran in Google colab.

Model Explanation & Observations

It is possible to predict the class value based on the original 6 attributes with really good accuracy.

However, this section requires us to predict the 'buying' attribute based on the other attributes and the class value. I would then try to reverse this, and use the 'buying' attribute as the class label to predict. But this in turn gives really bad accuracy after some testing.

Since we know that we CAN train based on the original attributes, we instead try out all buying labels for a particular entry, keeping the other attributes the same. We then get the predicted probabilities for the desired class value, and then get the largest probability and the respective buying label.

This will require more computations, since we are performing 4 predictions instead of just 1. This is the tradeoff that I can come up with at the moment, to get good performance with the model.

2 other decision trees will be shown. One that is trained and used in the same way, except that we include the 'persons' attribute and is more accurate.

The last decision tree is also trained to directly predict the 'buying' attribute, and its results are also shown here.

```
# imports
import os
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report

from sklearn.tree import DecisionTreeClassifier
```

```
# parameters related to the dataset
file_path = os.getcwd() + '/car.data'

col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', \
             'acceptability']

buying_labels = ['low', 'med', 'high', 'vhigh']

# labels for one-hot encoding
labels_buying = {'low': 0, 'med': 1, 'high': 2, 'vhigh': 3}
labels_maint = {'low': 0, 'med': 1, 'high': 2, 'vhigh': 3}
labels_doors = {'2': 0, '3': 1, '4': 2, '5more': 3}
labels_persons = {'2': 0, '4': 1, 'more': 2}
labels_lug_boot = {'small': 0, 'med': 1, 'big': 2}
labels_safety = {'low': 0, 'med': 1, 'high': 2}
labels_acceptability = {'unacc': 0, 'acc': 1, 'good': 2, 'vgood': 3}
```

```
# load the data
dataset_df = pd.read_csv(file_path, names = col_names)

# encode the data
dataset_df.iloc[:, 0].replace(labels_buying, inplace = True)
dataset_df.iloc[:, 1].replace(labels_maint, inplace = True)
dataset_df.iloc[:, 2].replace(labels_doors, inplace = True)
dataset_df.iloc[:, 3].replace(labels_persons, inplace = True)
dataset_df.iloc[:, 4].replace(labels_lug_boot, inplace = True)
dataset_df.iloc[:, 5].replace(labels_safety, inplace = True)

labels = dataset_df.iloc[:, 6]
features = dataset_df.iloc[:, :6]

# drop the persons column as it is unused
features.drop(columns = 'persons', inplace = True)

# train test split
x_train, x_test, y_train, y_test = train_test_split(features, \
                                                    labels, test_size = 0.3, \
                                                    random_state = 97)
```

Train the Model

We will train a decision tree classifier with the following inputs:

> Buying price
> Maintenance
> Number of doors
>
> Lug Boot Size
> Safety

And to predict the class value.

In this case, the performance is okay, partly because we are missing the 'person' attribute.

During predictions, predictions for all possible buying prices are used. The highest prediction confidence of the given class value is taken.

```
def create_and_train_model(x_train, y_train, x_test, y_test):
  """Uses SKLearn Grid Search CV to try out various parameters for a decision
  tree, and returns the best model trained.

  A performance report from SKLearn metrics will be produced, showing us the
  different metrics for each class label.
  """
  decision_tree = DecisionTreeClassifier()
  param_grid = {'criterion': ['gini', 'entropy'], \
                'max_depth': [5, 6, 7, 8, 9, 10, 11, 12], \
                'min_samples_leaf': [1, 2, 3], \
                'random_state':[97]}

  model = GridSearchCV(decision_tree, param_grid)
  model.fit(x_train, y_train)

  y_pred = model.predict(x_test)
  print('Report:')
  print(classification_report(y_test, y_pred))

  return model

def predict_buying_price(model, features: pd.DataFrame) -> list:
  """Takes in the following features, and provides a List of predictions for
  the buying price.

  Features:
    Maintenance
    Number of doors
    Lug Boot Size
    Safety
    Class value

  Performs a series of model predictions with different buying values, and takes
  the buying value with the highest probability of predicting the target class
  value.
  """
  buying_price_values = [0, 1, 2, 3]

  # re-arrange the data
  temp_features = features
  class_values = temp_features['acceptability']
  temp_features.drop(columns = 'acceptability', inplace = True)

  prediction_probabilities = [[] for i in range(len(features))]
  predictions = []
```

```
  # predict with all values of buying price
  for i in range(len(buying_price_values)):
    temp_features.insert(0, 'buying', [buying_price_values[i] for j in range(len(features)
    probabilities = model.predict_proba(temp_features)
    temp_features.drop(columns='buying', inplace=True)

    for j in range(len(probabilities)):
      prediction_probabilities[j].append(probabilities[j])

  # get the argmax of the class value, to get the appropriate buying price
  for i in range(len(prediction_probabilities)):
    correct_class_labels = [prediction_probabilities[i][j][class_values[i]] \
                            for j in range(4)]

    predictions.append(np.argmax(correct_class_labels))

  return predictions
```

```
# create and train the model
model = create_and_train_model(x_train, y_train, x_test, y_test)
```

```
    Report:
                  precision    recall  f1-score   support

             acc       0.60      0.80      0.68       119
            good       0.39      1.00      0.56        12
           unacc       0.92      0.77      0.84       373
           vgood       0.50      0.53      0.52        15

        accuracy                           0.78       519
       macro avg       0.60      0.78      0.65       519
    weighted avg       0.82      0.78      0.79       519
```

```
# run the input given
input_features = pd.DataFrame.from_dict({'maint': [2], 'doors': [2], \
                                         'lug_boot': [1], 'safety': [2], \
                                         'acceptability': [2]})

results = predict_buying_price(model, input_features)
for result in results:
  print(buying_labels[result])
```

```
    vhigh
```

Further Validation

To prove that this can work better, I will train a model with the 'persons' attribute, which produces an original model with higher performance metrics.

Since this needs the 'persons' field, I will provide the prediction for all 3 values of 'persons'.

```
# encode the data with the persons column
dataset_df.iloc[:, 0].replace(labels_buying, inplace = True)
dataset_df.iloc[:, 1].replace(labels_maint, inplace = True)
dataset_df.iloc[:, 2].replace(labels_doors, inplace = True)
dataset_df.iloc[:, 3].replace(labels_persons, inplace = True)
dataset_df.iloc[:, 4].replace(labels_lug_boot, inplace = True)
dataset_df.iloc[:, 5].replace(labels_safety, inplace = True)

labels = dataset_df.iloc[:, 6]
features = dataset_df.iloc[:, :6]

# train test split
x_train, x_test, y_train, y_test = train_test_split(features, \
                                                    labels, test_size = 0.3, \
                                                    random_state = 97)
```

```
# create and train the model
model = create_and_train_model(x_train, y_train, x_test, y_test)
```

```
     Report:
                   precision    recall  f1-score   support

              acc       0.95      0.95      0.95       119
             good       0.80      1.00      0.89        12
            unacc       0.99      0.99      0.99       373
            vgood       1.00      0.93      0.97        15

         accuracy                           0.98       519
        macro avg       0.94      0.97      0.95       519
     weighted avg       0.98      0.98      0.98       519
```

```
# get the predictions
input_features = pd.DataFrame.from_dict({'maint': [2, 2, 2], \
                                         'doors': [2, 2, 2], \
                                         'persons': [0, 1, 2], \
                                         'lug_boot': [1, 1, 1], \
                                         'safety': [2, 2, 2], \
                                         'acceptability': [2, 2, 2]})

results = predict_buying_price(model, input_features)

print('buying price if the persons field was: ')
print('2: ' + buying_labels[results[0]])
print('4: ' + buying_labels[results[1]])
print('more: ' + buying_labels[results[2]])
```

```
     buying price if the persons field was:
     2: low
     4: vhigh
```

```
more: vhigh
```

Original Decision Tree Classifier

This will show a decision tree if it was trained directly with the specific features, to predict the buying price.

```
# encode the data
dataset_df.iloc[:, 1].replace(labels_maint, inplace = True)
dataset_df.iloc[:, 2].replace(labels_doors, inplace = True)
dataset_df.iloc[:, 3].replace(labels_persons, inplace = True)
dataset_df.iloc[:, 4].replace(labels_lug_boot, inplace = True)
dataset_df.iloc[:, 5].replace(labels_safety, inplace = True)
dataset_df.iloc[:, 6].replace(labels_acceptability, inplace = True)

labels = dataset_df.iloc[:, 0]
features = dataset_df.iloc[:, 1:7]

# drop the persons column as it is unused
features.drop(columns = 'persons', inplace = True)

# train test split
x_train, x_test, y_train, y_test = train_test_split(features, \
                                                    labels, test_size = 0.3, \
                                                    random_state = 97)
```

```
# create and train the model
model = create_and_train_model(x_train, y_train, x_test, y_test)
```

```
Report:
              precision    recall  f1-score   support

           0       0.28      0.17      0.21       139
           1       0.21      0.11      0.14       132
           2       0.27      0.19      0.23       129
           3       0.24      0.55      0.33       119

    accuracy                           0.25       519
   macro avg       0.25      0.25      0.23       519
weighted avg       0.25      0.25      0.23       519
```

```
# and then perform the prediction
input_features = pd.DataFrame.from_dict({'maint': [2], 'doors': [2], \
                                         'lug_boot': [1], 'safety': [2], \
                                         'acceptability': [2]})

results = model.predict(input_features)
for result in results:
  print(buying_labels[result])
```

```
low
```

✓ 0s completed at 5:13 PM