Decision Tree for Engine Type Classification

1. Load the complete data file and drop the columns that contain a single value, if any

In [1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

file_name = "quality_engine_anon.csv"
#file_name = "quality_engine_anon_err.csv"
nq = pd.read_csv(file_name, index_col="Unnamed: 0", encoding='utf-8')

print('Dataset has ' + str(nq.columns.size) + ' columns')
# check if there are any columns that are all filled with the same symbol
# and eventually drop them
to_drop = []
for i in nq.columns:
    if len(set(nq[i])) == 1:
        to_drop.append(i)

nq.drop(to_drop,axis=1,inplace=True)
ldf = nq.copy()
ldf.head()
```

```
Dataset has 4019 columns
```

Out[1]:

	ENGTYPE	FEAT_1	FEAT_2	FEAT_3	FEAT_4	FEAT_5	FEAT_6	FEAT_
TEST_FILE_0	ENGINE_T2	a	d	a	a	d	a	d
TEST_FILE_1	ENGINE_T2	a	d	a	a	d	a	d
TEST_FILE_2	ENGINE_T2	a	d	a	a	d	a	d
TEST_FILE_3	ENGINE_T2	a	d	a	a	d	a	d
TEST_FILE_4	ENGINE_T2	a	d	a	a	d	a	d

5 rows × 4019 columns

2. Replace all categorical values with numerical ones - no one-hot mapping

The values in each feature columns have only a qualitative meaning and are related to the real test values as follows:

- a. all values are numerical
- b. some values are numerical but not all of them are present
- c. textual values or overflows
- d. not present in the test file

In [11]:

```
# replace a, b, c, d with numerical values
ldf[ldf=="a"] = 0
ldf[ldf=="b"] = 1
ldf[ldf=="c"] = 2
ldf[ldf=="d"] = 3

pred_eng= list(set(ldf['ENGTYPE']))
pred_eng.sort()
print(pred_eng)
map_to_int = {name: n for n, name in enumerate(pred_eng)}

ldf['ENGTYPE'] = ldf['ENGTYPE'].replace(map_to_int) # true values
ldf.head()
```

```
['ENGINE_T0', 'ENGINE_T1', 'ENGINE_T2', 'ENGINE_T3']
```

Out[11]:

	ENGTYPE	FEAT_1	FEAT_2	FEAT_3	FEAT_4	FEAT_5	FEAT_6	FEAT_7
TEST_FILE_0	0	0	3	0	0	3	0	3
TEST_FILE_1	2	0	3	0	0	3	0	3
TEST_FILE_2	2	0	3	0	0	3	0	3
TEST_FILE_3	2	0	3	0	0	3	0	3
TEST_FILE_4	2	0	3	0	0	3	0	3

5 rows × 4019 columns

In [13]:
```
# OPTIONAL reorder columns so that ENGTYPE is in the first position
lc = list(ldf.columns)
lc.remove('ENGTYPE')
lc.insert(0,'ENGTYPE')
ldf = ldf[lc]
ldf.head()
```

Out[13]:

	ENGTYPE	FEAT_1	FEAT_2	FEAT_3	FEAT_4	FEAT_5	FEAT_6	FEAT_7
TEST_FILE_0	0	0	3	0	0	3	0	3
TEST_FILE_1	2	0	3	0	0	3	0	3
TEST_FILE_2	2	0	3	0	0	3	0	3
TEST_FILE_3	2	0	3	0	0	3	0	3
TEST_FILE_4	2	0	3	0	0	3	0	3

5 rows × 4019 columns

3. OPTIONAL trains a new Sklearn decision tree for the current dataset and saves the decision tree onto a file

This allows to produce a new decision tree to implement the classification routines at sections 4x in case of new datasets

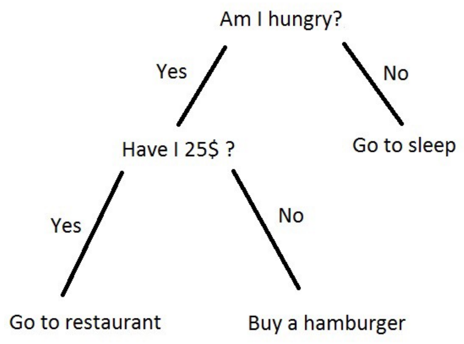
---

What is a decision tree?

It's a decision support tool based on a tree-like model.
Decision tree learning uses a decision tree as predictive model that can be used to visually represent the decision making process. The goal is to create a model that predicts the value of a target variable based on several input variables.
The basic components are nodes, branches and leaves.

- Each **node** corresponds to testing an input variable
- Each **branch** represents a possible value for the input variable
- Each **leaf** represents a value for the target variable

A tree can be learned by recursively splitting the data set into subsets based on a test on an input variable until all items in the subset have the same class value or no further splitting is possible.

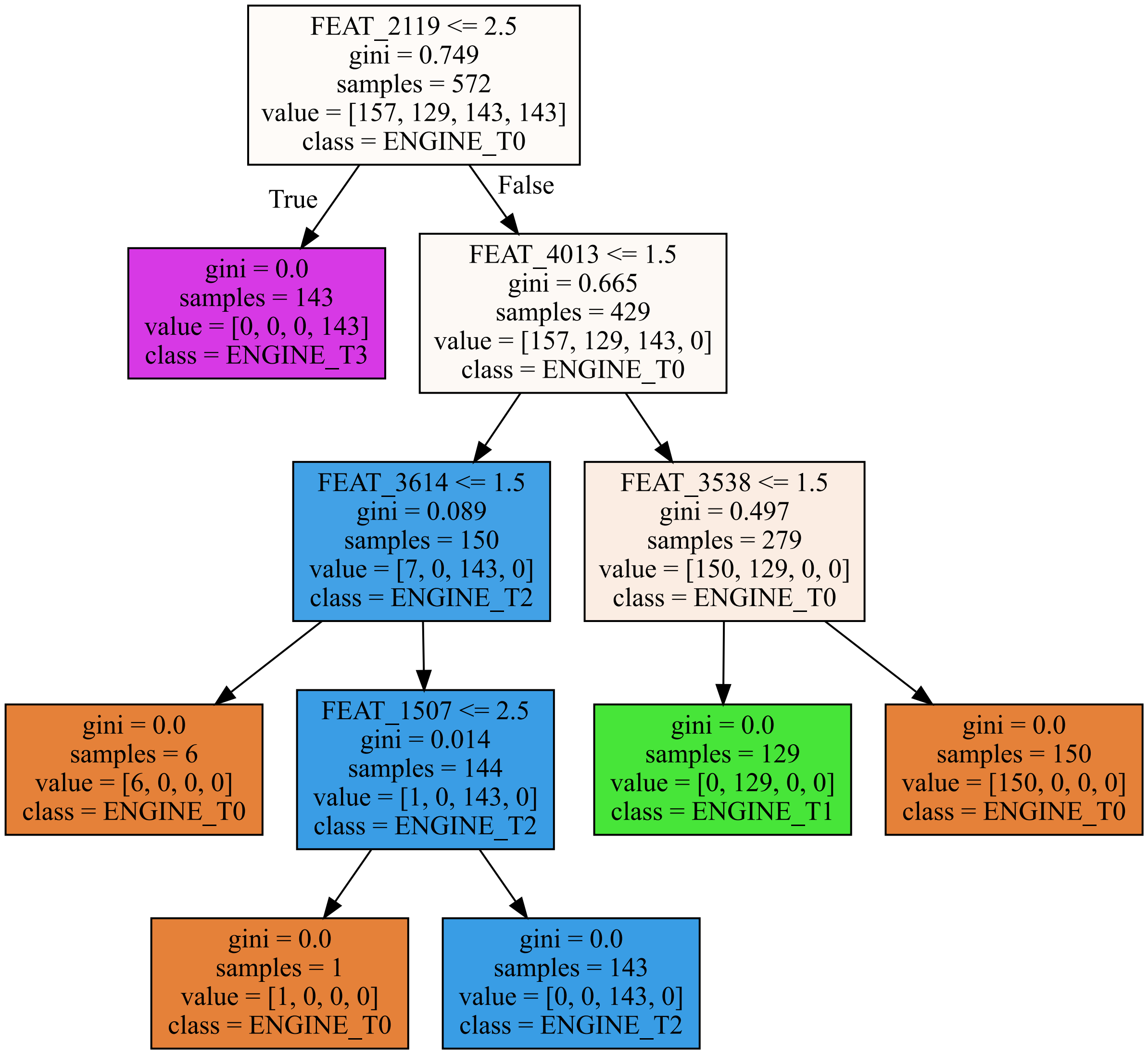


```
In [12]: from sklearn.tree import DecisionTreeClassifier
         from sklearn.model_selection import cross_val_score
         from sklearn import tree

         clf = DecisionTreeClassifier(random_state=0)
         tscore=cross_val_score(clf, ldf.iloc[:,1:],ldf.iloc[:,0], cv=10)
         print(np.mean(tscore))

         # Create DOT data
         clf.fit(ldf.iloc[:,1:],ldf.iloc[:,0])

         dotfile='dtree_tasha.dot'
         tree.export_graphviz(clf, feature_names=ldf.columns[1:], class_names=pred_eng,
          out_file=dotfile)
```

```
0.9964869029275809
```

And optionally plot the decision tree:

In [13]:
```
from sklearn import tree
from IPython.display import SVG
from graphviz import Source
from IPython.display import display

# print dataset description
graph = Source(tree.export_graphviz(clf, out_file=None,feature_names=ldf.colum
ns[1:],class_names=pred_eng,filled = True))
display(SVG(graph.pipe(format='svg')))
```



4. Setup the decision tree defined by Sklearn Decision Tree Classifier

In [14]:
```
# implement Decision Tree rules from Sklearn DecisionTreeClassifier
def tree_predict(f,ldf):
    a = 0   # numerical values
    b = 1   # some numerical values but not all of them
    c = 2   # all values are string values or overflow values
    d = 3   # variable doesn't exist in the datafile
    pred_eng = {'ENGINE_T0':0,'ENGINE_T1':1,'ENGINE_T2':2,'ENGINE_T3':3}

    v = ldf.loc[f]['FEAT_2119']
    if v <= 2.5:
        r = pred_eng['ENGINE_T3']
    else:
        v = ldf.loc[f]['FEAT_4013']
        if v <= 1.5:
            v = ldf.loc[f]['FEAT_3614']
            if v <= 1.5:
                r = pred_eng['ENGINE_T0']
            else:
                r = pred_eng['ENGINE_T2']
        else:
            v = ldf.loc[f]['FEAT_3629']
            if v <= 1.5:
                r = pred_eng['ENGINE_T0']
            else:
                r = pred_eng['ENGINE_T1']

    return r
```

5. Execute the classification task

In [15]:
```
# apply Decision Tree rules
engines = dict()
for f in ldf.index:
    r = tree_predict(f,ldf)
    engines[f] = r

# This is the predicted engine types list, to be compared with the actual one
edf = pd.DataFrame.from_dict(engines,orient='index',columns=['ENGTYPE'])
edf.head()
```

Out[15]:

	ENGTYPE
TEST_FILE_0	2
TEST_FILE_1	2
TEST_FILE_2	2
TEST_FILE_3	2
TEST_FILE_4	2

6. Evaluate classification accuracy

In [16]:
```
good = sum(edf['ENGTYPE'] == ldf['ENGTYPE'])
count = ldf.shape[0]
accuracy = good/count
print('Accuracy: %f' %(accuracy))
```

```
Accuracy: 0.998252
```

7. Identify errors, if accuracy is lower than 100%

In [17]:
```
# OPTIONAL identify the erroneously classified engine and its source file
ixlist = np.argwhere(edf['ENGTYPE'] != ldf['ENGTYPE'])
ixlist = ixlist.ravel()
if ixlist.size:
    for ix in ixlist:
        te = pred_eng[ldf.iloc[ix]['ENGTYPE']]
        me = pred_eng[edf.iloc[ix]['ENGTYPE']]
        fn = ldf.index[ix]
        print("Engine %s has been misclassified into %s in file %s" %(te,me,fn
))
else:
    print('No misclassified engines')
```

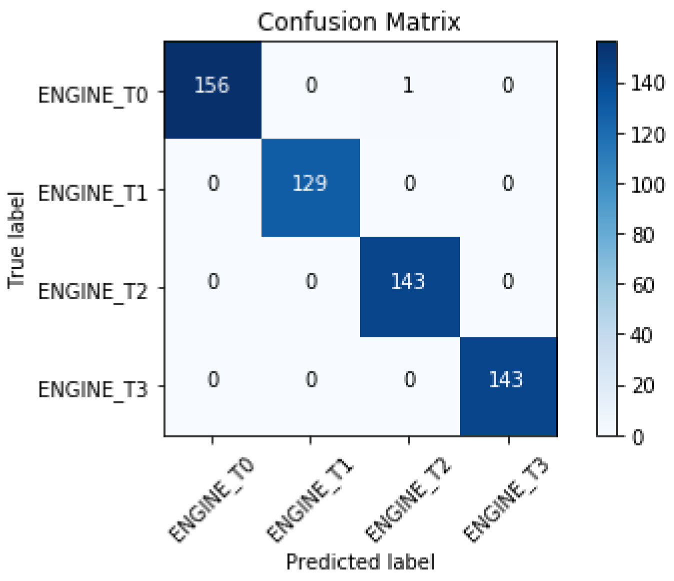
```
Engine ENGINE_T0 has been misclassified into ENGINE_T2 in file TEST_FILE_0
```

Comments

The approach identified can be described as follows:

1. load the dataset and replace a, b, c, d class values with 0, 1, 2, 3 respectively
2. replace target engine types through an integer enumerator
3. train a decision tree classifier and save the new decision tree for future visualization
4. define the engine classifier according to the plot of the decision tree
5. feed the dataset to the tree algorithm and check prediction for accuracy

In [18]:
```
import itertools
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(ldf.iloc[:]['ENGTYPE'],edf.iloc[:]['ENGTYPE'])

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        #print("Normalized confusion matrix")
    #else:
        #print('Confusion matrix, without normalization')

    #print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

plt.figure()
plot_confusion_matrix(cm, classes=pred_eng, normalize=False,
                      title='Confusion Matrix')

plt.show()
```


Confusion Matrix
True label
Predicted label
ENGINE_T0
ENGINE_T1
ENGINE_T2
ENGINE_T3
156
0
1
0
0
129
0
0
0
0
143
0
0
0
0
143
ENGINE_T0
ENGINE_T1
ENGINE_T2
ENGINE_T3
140
120
100
80
60
40
20
0