# Project_3

July 28, 2020

## 1 Project 3. Cars datase analysis

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from os import walk
import re
```

### 1.1 1. Read files

[2]:
```
data_path = './data'
```

[3]:
```
for root, dirs, files in walk(data_path):
    all_files = [data_path+'/'+file for file in files if file.split('.')[-1] in␣
 ↪['data', 'names']]

for file_num in range(len(all_files)):
    print(f'#{file_num} ->', all_files[file_num])
```

```
#0 -> ./data/car.names
#1 -> ./data/car.data
```

[4]:
```
# in here we choose the index from above result. for example here we want data␣
 ↪file with index 2
# and data names file with index 3

# select index number for names file and data file (it's manual cuz we can use␣
 ↪it later for
# other projects).
# ----------------------
names_index = 0
data_index = 1
# ----------------------

# open names file
```

```
names_file = [line.strip() for line in open(all_files[names_index], 'r').
 ↪readlines()]

print_flag = False
attrs = []
for line in names_file:
    if 'Missing Attribute' in line:
        break

    if print_flag:
        attr = line.split(' ')

        if len(attr) > 0 and len(attr[0]) > 0:
            attrs.append(attr[0])

    if 'Attribute Values' in line:
        print_flag = True

attrs.append('class')
# print all attributes
print('\nattributes count: ', len(attrs), '\n'*2)
print(attrs)
```

```
attributes count:  7


['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

[5]:
```
data_df = pd.read_csv(all_files[data_index])
data_df.columns = attrs
data_df
```

[5]:

| | 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 |
| … | … | … | … | … | … | … | … |
| 1722 | low | low | 5more | more | med | med | good |
| 1723 | low | low | 5more | more | med | high | vgood |
| 1724 | low | low | 5more | more | big | low | unacc |
| 1725 | low | low | 5more | more | big | med | good |
| 1726 | low | low | 5more | more | big | high | vgood |

[1727 rows x 7 columns]

[6]:
```
from sklearn.model_selection import train_test_split

y = pd.DataFrame(data_df['class'])
X = data_df.drop(['class'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,␣
↪test_size=0.3, random_state=25)
```

### 1.2 2. Train & test splitting and Label incoding

[7]:
```
# Get list of categorical variables
s = (X_train.dtypes == 'object')
object_cols = list(s[s].index)

print("Categorical variables:")
print(object_cols)
print(len(object_cols))
# LOL!! all of them are object types!
```

```
Categorical variables:
['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety']
6
```

[8]:
```
from sklearn.preprocessing import LabelEncoder

# Make copy to avoid changing original data
label_X_train = X_train.copy()
label_X_test = X_test.copy()

label_y_train = y_train.copy()
label_y_test = y_test.copy()

# Apply label encoder to each column with categorical data
label_encoder = LabelEncoder()
for col in object_cols:
    label_X_train[col] = label_encoder.fit_transform(X_train[col])
    label_X_test[col] = label_encoder.transform(X_test[col])

X_train = label_X_train
X_test = label_X_test

for col in y_train.columns:
    label_y_train[col] = label_encoder.fit_transform(y_train[col])
    label_y_test[col] = label_encoder.transform(y_test[col])

y_train = label_y_train
```

```
y_test = label_y_test
```

### 1.3 3.1. Decision Tree model

[9]:
```
from sklearn.tree import DecisionTreeClassifier

tree_model = DecisionTreeClassifier(random_state=25)
tree_model.fit(X_train, y_train)
```

[9]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=25, splitter='best')
```

[10]:
```
from sklearn.metrics import classification_report

tree_prediction = tree_model.predict(X_test)
print(classification_report(y_test, tree_prediction))
```

```
              precision    recall  f1-score   support

           0       0.98      0.91      0.94       117
           1       0.88      0.95      0.91        22
           2       0.98      0.99      0.99       362
           3       0.95      1.00      0.97        18

    accuracy                           0.97       519
   macro avg       0.95      0.96      0.95       519
weighted avg       0.97      0.97      0.97       519
```

### 1.4 3.2. Naïve Bayes model

[11]:
```
from sklearn.naive_bayes import MultinomialNB

NB_model = MultinomialNB()
NB_model.fit(X_train, y_train)
```

```
/home/hakim/.local/lib/python3.6/site-packages/sklearn/utils/validation.py:760:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[11]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

[12]:
```
from sklearn.metrics import classification_report

NB_prediction = NB_model.predict(X_test)
print(classification_report(y_test, NB_prediction))
```

```
              precision    recall  f1-score   support

           0       0.50      0.01      0.02       117
           1       0.00      0.00      0.00        22
           2       0.70      1.00      0.82       362
           3       0.00      0.00      0.00        18

    accuracy                           0.70       519
   macro avg       0.30      0.25      0.21       519
weighted avg       0.60      0.70      0.58       519


/home/hakim/.local/lib/python3.6/site-
packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

### 1.5 3.3. Neural Networks model

[13]:
```
from sklearn.neural_network import MLPClassifier
cnn_model = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5,␣
 ↪2), random_state=1)
cnn_model.fit(X_train, y_train)

cnn_prediction = cnn_model.predict(X_test)
```

```
/home/hakim/.local/lib/python3.6/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:934:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/home/hakim/.local/lib/python3.6/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:470:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

[14]:
```
from sklearn.metrics import classification_report

cnn_prediction = cnn_model.predict(X_test)
print(classification_report(y_test, cnn_prediction))
```

```
              precision    recall  f1-score   support

           0       0.79      0.65      0.71       117
           1       0.72      0.59      0.65        22
           2       0.88      0.96      0.92       362
           3       0.30      0.17      0.21        18

    accuracy                           0.85       519
   macro avg       0.67      0.59      0.62       519
weighted avg       0.84      0.85      0.84       519
```