random-forest-classifier-tutorial

August 8, 2023

1 Car Evaluation with Random-forest-classifier Algorithm

Welcome to the Random Forest Classification tutorial using the Car Evaluation Dataset! In this tutorial, we will explore the fascinating world of machine learning and delve into the Random Forest algorithm.

The Car Evaluation Database presents an exciting opportunity to understand how cars are evaluated based on attributes such as buying price, maintenance cost, number of doors, passenger capacity, luggage boot size, and safety rating.

By the end of this tutorial, you'll gain insights into the fundamentals of Random Forest classification and learn how to make predictions about car evaluation outcomes and all while gaining a strong foundation in classification with scikit-learn.

2 Introduction to Random Forest algorithm

Random forest is a supervised learning algorithm. It has two variations – one is used for classification problems and other is used for regression problems.

It is one of the most flexible and easy to use algorithm.

Random forest algorithm combines multiple decision-trees, resulting in a forest of trees, hence the name `Random Forest`. In the random forest classifier, the higher the number of trees in the forest results in higher accuracy.

It creates decision trees on the given data samples, gets prediction from each tree and selects the best solution by means of voting.

It is also a pretty good indicator of feature importance.

3 Difference between Random Forests and Decision Trees

I will compare random forests with decision-trees. Some salient features of comparison are as follows:-

1. Random forests is a set of multiple decision-trees.
2. Decision-trees are computationally faster as compared to random forests.
3. Deep decision-trees may suffer from overfitting. Random forest prevents overfitting by creating trees on random forests.

4. Random forest is difficult to interpret. But, a decision-tree is easily interpretable and can be converted to rules.

4 Dataset Description:

Car Evaluation Database was derived from a simple hierarchical decision model originally developed for the demonstration of DEX, M. Bohanec, V. Rajkovic: Expert system for decision making. Sistemica 1(1), pp. 145-157, 1990.). The model evaluates cars according to the following concept structure: 1. CAR car acceptability 2. PRICE overall price 3. buying buying price 4. maint price of the maintenance 5. TECH technical characteristics 6. COMFORT comfort 7. doors number of doors 8. persons capacity in terms of persons to carry 9. lug_boot the size of luggage boot 10. safety estimated safety of the car

Input attributes are printed in lowercase. Besides the target concept (CAR), the model includes three intermediate concepts: PRICE, TECH, and COMFORT. .

The Car Evaluation Database contains examples with the structural information removed, i.e., directly relates CAR to the six input attributes: buying, maint, doors, persons, lug_boot, and safety.

Because of known underlying concept structure, this database may be particularly useful for testing constructive induction and structure discovery methods.

Attribute Information:

1. buying: vhigh, high, med, low.
2. maint: vhigh, high, med, low.
3. doors: 2, 3, 4, 5more.
4. persons: 2, 4, more.
5. lug_boot: small, med, big.
6. safety: low, med, high.

5 Import libraries

[1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # data visualization
import seaborn as sns # statistical data visualization
%matplotlib inline
```

6 Import dataset

[2]:
```
df = pd.read_csv('car_evaluation.csv')
df
```

[2]:
```
      vhigh vhigh.1     2   2.1  small   low  unacc
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]
```

7 Exploratory Data Analysis(EDA)

[3]:
```
# view dimensions of dataset
df.shape
```

[3]:
```
(1727, 7)
```

We can see that there are 1728 instances and 7 variables in the data set.

7.0.1 View top 5 rows of dataset

[4]:
```
# preview the dataset
df.head()
```

[4]:
```
   vhigh vhigh.1  2 2.1  small   low  unacc
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
```

7.0.2 Rename column names

We can see that the dataset does not have proper column names. The columns are merely labelled as 0,1,2…. and so on. We should give proper names to the columns. I will do it as follows:-

[5]:
```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety',␣
 ↪'class']
df.columns = col_names
col_names
```

[5]:
```
['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

```
[6]: # let's again preview the dataset
     df.head()
```

```
[6]:   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
```

We can see that the column names are renamed. Now, the columns have meaningful names.

7.0.3 View summary of dataset

```
[7]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1727 entries, 0 to 1726
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   buying    1727 non-null   object
 1   maint     1727 non-null   object
 2   doors     1727 non-null   object
 3   persons   1727 non-null   object
 4   lug_boot  1727 non-null   object
 5   safety    1727 non-null   object
 6   class     1727 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

7.0.4 Frequency distribution of values in variables

Now, I will check the frequency counts of categorical variables.

```
[8]: col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety',␣
      ↪'class']

     for col in col_names:
         print(df[col].value_counts())
```

```
high     432
med      432
low      432
vhigh    431
Name: buying, dtype: int64
high     432
med      432
```

```
low       432
vhigh     431
Name: maint, dtype: int64
3         432
4         432
5more     432
2         431
Name: doors, dtype: int64
4        576
more     576
2        575
Name: persons, dtype: int64
med       576
big       576
small     575
Name: lug_boot, dtype: int64
med      576
high     576
low      575
Name: safety, dtype: int64
unacc     1209
acc        384
good        69
vgood       65
Name: class, dtype: int64
```

We can see that the `doors` and `persons` are categorical in nature. So, I will treat them as categorical variables.

7.0.5 Summary of variables

- There are 7 variables in the dataset. All the variables are of categorical data type.
- These are given by `buying`, `maint`, `doors`, `persons`, `lug_boot`, `safety` and `class`.
- `class` is the target variable.

7.0.6 Explore class variable

[9]:
```
df['class'].value_counts()
```

[9]:
```
unacc     1209
acc        384
good        69
vgood       65
Name: class, dtype: int64
```

The `class` target variable is ordinal in nature.

7.0.7 Missing values in variables

```
[10]: # check missing values in variables

      df.isnull().sum()
```

```
[10]: buying      0
      maint       0
      doors       0
      persons     0
      lug_boot    0
      safety      0
      class       0
      dtype: int64
```

We can see that there are no missing values in the dataset. I have checked the frequency distribution of values previously. It also confirms that there are no missing values in the dataset.

8 Declare feature vector and target variable

When training a Random Forest model, you provide the algorithm with the feature vectors (input data) and their corresponding target variable (output or labels).

The model learns the patterns and relationships between the features and the target variable during the training process.

Once trained, the Random Forest model can take new feature vectors (unseen data) as input and predict the corresponding target variable or make classifications based on what it has learned during training.

```
[11]: X = df.drop(['class'], axis=1)
      y = df['class']
```

9 Split data into separate training and test set

```
[12]: # split data into training and testing sets
      from sklearn.model_selection import train_test_split

      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33,␣
        ↪random_state = 42)
```

```
[13]: # check the shape of X_train and X_test
      X_train.shape, X_test.shape
```

```
[13]: ((1157, 6), (570, 6))
```

10 Feature Engineering

Feature Engineering is the process of transforming raw data into useful features that help us to understand our model better and increase its predictive power.

First,Check the data types of variables again.

[14]:
```
# check data types in X_train
X_train.dtypes
```

[14]:
```
buying      object
maint       object
doors       object
persons     object
lug_boot    object
safety      object
dtype: object
```

In scikit-learn's Random Forest algorithm, encoding categorical variables is a necessary step before training the model. Random Forest, like most machine learning algorithms, requires that all input data be in numerical format. Categorical variables, which represent categories or labels (e.g., colors, types, classes), need to be converted into numerical values.

10.1 Purpose of Encoding Categorical Variables:

Numerical Representation: Random Forest algorithm can only work with numerical data. Encoding categorical variables helps convert these non-numeric categories into numerical representations that the algorithm can understand.

Feature Inclusion: By encoding categorical variables, you ensure that all features in the dataset can be used during model training. If categorical variables are not encoded, they might be ignored by the algorithm, leading to valuable information loss.

[15]:
```
X_train.head()
```

[15]:
```
     buying  maint  doors persons lug_boot safety
83    vhigh  vhigh  5more       2      med    low
48    vhigh  vhigh      3    more      med    med
468    high  vhigh      3       4    small    med
155   vhigh   high      3    more      med    low
1043    med   high      4    more    small    low
```

10.2 Category encoders

Category encoders is a Python library that specializes in encoding categorical variables for machine learning tasks. It offers a range of techniques to convert categorical data into numerical format, making it compatible with various machine learning algorithms.

Category encoders provides different encoding methods, such as Ordinal Encoding, One-Hot Encoding, Target Encoding, and more.

It is particularly useful when dealing with datasets that contain categorical variables and when the scikit-learn's default encoding methods may not be sufficient.

We can see that all the variables are ordinal categorical data type.

[16]:
```
# import category encoders

import category_encoders as ce
from sklearn.preprocessing import OrdinalEncoder
```

[17]:
```
import category_encoders as ce

encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'doors', 'persons',␣
 ↪'lug_boot', 'safety'])

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

[18]:
```
X_train.head()
```

[18]:
```
      buying  maint  doors  persons  lug_boot  safety
83         1      1      1        1         1       1
48         1      1      2        2         1       2
468        2      1      2        3         2       2
155        1      2      2        2         1       1
1043       3      2      3        2         2       1
```

[19]:
```
X_test.head()
```

[19]:
```
      buying  maint  doors  persons  lug_boot  safety
599        2      2      3        1         3       1
932        3      1      3        3         3       1
628        2      2      1        1         3       3
1497       4      2      1        3         1       2
1262       3      4      3        2         1       1
```

We now have training and test set ready for model building.

11 Random Forest Classifier model with default parameters

The Random Forest Classifier is an ensemble learning algorithm that combines multiple decision trees to make accurate predictions.

Each decision tree is built on a random subset of the training data and a random subset of features.

The default parameters of the Random Forest Classifier are carefully chosen to work well in a variety of scenarios without requiring extensive tuning.

Random state

The random_state parameter is used in machine learning algorithms and functions to introduce an element of randomness or to control the randomness within the algorithm.

It's particularly helpful for reproducibility and ensuring that your results can be recreated even if the algorithm involves random processes.

Here, I have build the Random Forest Classifier model with default parameter of `n_estimators = 10`. So, I have used 10 decision-trees to build the model. Now, I will increase the number of decision-trees and see its effect on accuracy

[20]:
```
# import Random Forest classifier

from sklearn.ensemble import RandomForestClassifier

# instantiate the classifier
rfc = RandomForestClassifier(n_estimators=10, random_state=0)



# fit the model
rfc.fit(X_train, y_train)

# Predict the Test set results
y_pred = rfc.predict(X_test)

# Check accuracy score
from sklearn.metrics import accuracy_score
print('Model accuracy score with 10 decision-trees : {0:0.4f}'.␣
 ↪format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with 10 decision-trees : 0.9474
```

Here, **y__test** are the true class labels and **y__pred** are the predicted class labels in the test-set.

12 Random Forest Classifier model with parameter n__estimators=100

[21]:
```
# instantiate the classifier with n_estimators = 100
rfc_100 = RandomForestClassifier(n_estimators=100, random_state=0)



# fit the model to the training set
rfc_100.fit(X_train, y_train)

# Predict on the test set results
y_pred_100 = rfc_100.predict(X_test)

# Check accuracy score
```

```
print('Model accuracy score with 100 decision-trees : {0:0.4f}'.
 ↪format(accuracy_score(y_test, y_pred_100)))
```

```
Model accuracy score with 100 decision-trees : 0.9649
```

The model accuracy score with 10 decision-trees is 0.9316 but the same with 100 decision-trees is 0.9649. So, as expected accuracy increases with number of decision-trees in the model.

13 Feature selection with Random Forests

Random forests algorithm can be used for feature selection process. This algorithm can be used to rank the importance of variables in a regression or classification problem.

We measure the variable importance in a dataset by fitting the random forest algorithm to the data. During the fitting process, the out-of-bag error for each data point is recorded and averaged over the forest.

Features which produce large values for this score are ranked as more important than features which produce small values. Based on this score, we will choose the most important features and drop the least important ones for model building.

14 Find important features with Random Forest model

Until now, I have used all the features given in the model. Now, I will select only the important features, build the model using these features and see its effect on accuracy.

First, I will create the Random Forest model as follows:-

[22]:
```
# create the classifier with n_estimators = 100
clf = RandomForestClassifier(n_estimators=100, random_state=0)

# fit the model to the training set
clf.fit(X_train, y_train)
```

[22]:
```
RandomForestClassifier(random_state=0)
```

Now, I will use the feature importance variable to see feature importance scores.

[23]:
```
# view the feature scores

feature_scores = pd.Series(clf.feature_importances_, index=X_train.columns).
 ↪sort_values(ascending=False)
feature_scores
```

[23]:
```
safety      0.291657
persons     0.235380
buying      0.160692
maint       0.134143
lug_boot    0.111595
```

```
doors       0.066533
dtype: float64
```

We can see that the most important feature is `safety` and least important feature is `doors`.

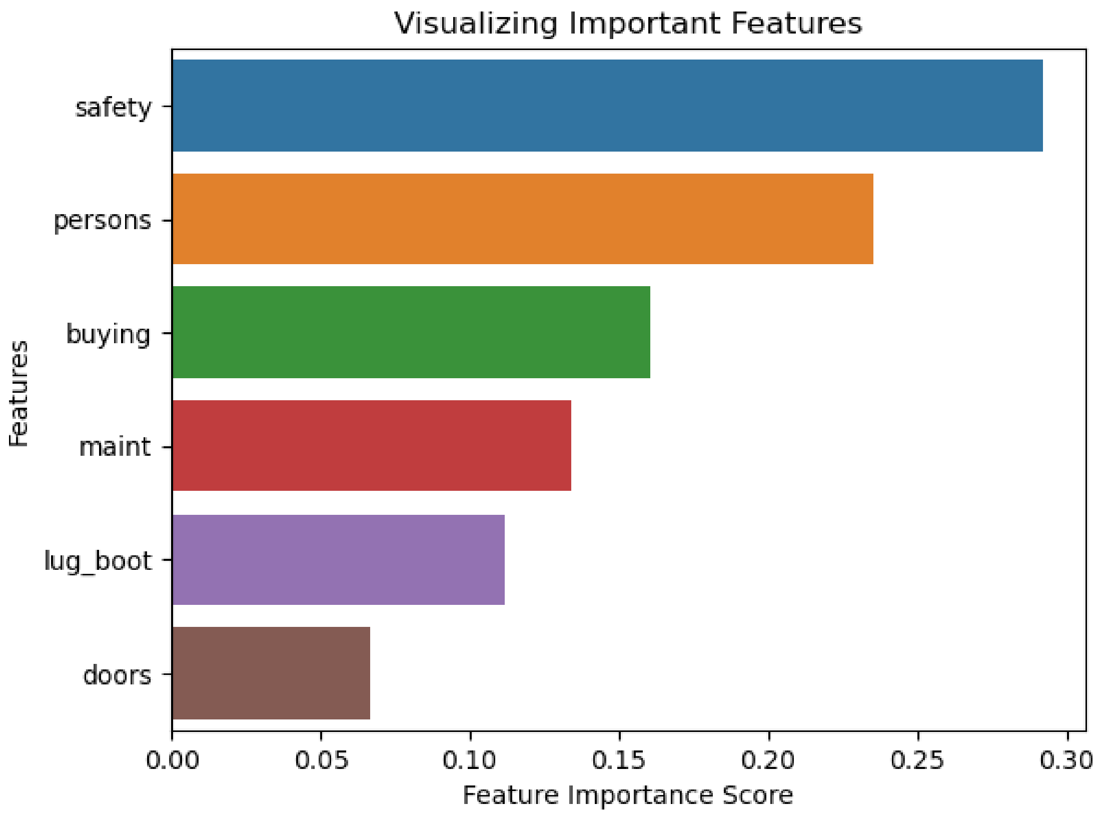
15 Visualize feature scores of the features

[24]:
```
# Creating a seaborn bar plot
sns.barplot(x=feature_scores, y=feature_scores.index)

# Add labels to the graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')

# Add title to the graph
plt.title("Visualizing Important Features")

# Visualize the graph
plt.show()
```



16 Build Random Forest model on selected features

Now, I will drop the least important feature `doors` from the model, rebuild the model and check its effect on accuracy.

```
[25]: # declare feature vector and target variable
X = df.drop(['class', 'doors'], axis=1)
y = df['class']
```

```
[26]: # split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33,␣
 ↪random_state = 42)
```

Now, I will build the random forest model and check accuracy.

```
[27]: # encode categorical variables with ordinal encoding
encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'persons', 'lug_boot',␣
 ↪'safety'])


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

```
[28]: # instantiate the classifier with n_estimators = 100
clf = RandomForestClassifier(random_state=0)

# fit the model to the training set
clf.fit(X_train, y_train)



# Predict on the test set results
y_pred = clf.predict(X_test)

# Check accuracy score
print('Model accuracy score with doors variable removed : {0:0.4f}'.␣
 ↪format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with doors variable removed : 0.9263
```

I have removed the `doors` variable from the model, rebuild it and checked its accuracy. The accuracy of the model with `doors` variable removed is 0.9264. The accuracy of the model with all the variables taken into account is 0.9263. So, we can see that the model accuracy has been improved with `doors` variable removed from the model.

17 Confusion matrix

A confusion matrix is a tool for summarizing the performance of a classification algorithm. A confusion matrix will give us a clear picture of classification model performance and the types of errors produced by the model. It gives us a summary of correct and incorrect predictions broken down by each category. The summary is represented in a tabular form.

Four types of outcomes are possible while evaluating a classification model performance. These four outcomes are described below:-

True Positives (TP) – True Positives occur when we predict an observation belongs to a certain class and the observation actually belongs to that class.

True Negatives (TN) – True Negatives occur when we predict an observation does not belong to a certain class and the observation actually does not belong to that class.

False Positives (FP) – False Positives occur when we predict an observation belongs to a certain class but the observation actually does not belong to that class. This type of error is called **Type I error.**

False Negatives (FN) – False Negatives occur when we predict an observation does not belong to a certain class but the observation actually belongs to that class. This is a very serious error and it is called **Type II error.**

These four outcomes are summarized in a confusion matrix given below.

[29]:
```python
# Print the Confusion Matrix and slice it into four pieces
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
print('Confusion matrix\n\n', cm)
```

```
Confusion matrix

 [[108   5  12   2]
 [  1  10   2   5]
 [ 10   0 389   0]
 [  4   1   0  21]]
```

18 Classification Report

Classification report is another way to evaluate the classification model performance. It displays the **precision**, **recall**, **f1** and **support** scores for the model.

We can print a classification report as follows:-

[30]:
```python
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         acc       0.88      0.85      0.86       127
```

```
        good       0.62      0.56      0.59        18
       unacc       0.97      0.97      0.97       399
       vgood       0.75      0.81      0.78        26

    accuracy                           0.93       570
   macro avg       0.80      0.80      0.80       570
weighted avg       0.93      0.93      0.93       570
```

19 Conclusion

In conclusion, the utilization of the Random Forest classification algorithm on the Car Evaluation Database, which encompasses attributes such as buying, maintenance, number of doors, seating capacity, luggage capacity, and safety rating, offers a robust approach to predict car evaluations. By harnessing the power of ensemble learning through Random Forest, we can effectively classify cars into distinct categories like unacc (unacceptable), acc (acceptable), good, and vgood (very good), facilitating informed decision-making in the domain of car assessment. This approach capitalizes on the inherent concept structure present in the data, making it particularly suitable for constructive induction and structure discovery methods.

19.1 Thank You!

```
[ ]:
```