2. DT - Risky Bank Loan

November 28, 2022

1 Machine Learning Project Lifecycle

Step 1: Exploratory Data Analysis

Step 2: Feature engineering and selection

Step 3: Data preparation for modelling (train/test split)

Step 4: Model Building

Step 5: Model Validation & Evaluation

Step 6: Model Tuning

Step 9: Model Deployment

2 Decision Tree - Example

Problem: Predicting risky bank loans using Decision Trees

The default vector indicates whether the loan applicant was unable to meet the agreed payment terms and went into default. A total of 30 percent of the loans in this dataset went into default. We have to train our model and predict such defaulters.

Data: 1. checking_balance - object

2. months_loan_duration - int64
3. credit_history - object
4. purpose - object
5. amount - int64
6. savings_balance - object
7. employment_length - object
8. installment_rate - int64
9. personal_status - object
10. other_debtors - object
11. residence_history - int64
12. property - object

13. age - int64
14. installment_plan - object
15. housing - object
16. existing_credits - int64
17. job - object
18. dependents - int64
19. telephone - object
20. foreign_worker - object
21. default - int64 (Target variable/Label)
    a) default = 1 –> Normal Customer
    b) default = 2 –> Risky Customer/Probable Deaulter

3 Load the necessary packages

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import confusion_matrix, accuracy_score
#pd.set_option('display.max_columns',30)
```

4 Exploring the data

[ ]:
```
df = pd.read_csv('../data/credit.csv')
```

[ ]:
```
df.head()
```

[ ]:
```
df.dtypes
```

[ ]:
```
df.describe()
```

[ ]:
```
df.isnull().sum() #checking NA values
```

[ ]:
```
df.checking_balance.value_counts()
```

[ ]:
```
df['savings_balance'].value_counts()
```

5 Data preparation

5.1 Find out the columns which are strings and cateogrical

- Checking unique values in each column to find the categorical columns.
- The description of data tells us which columns are categorical and which are continous.

[ ]:
```
# Checking unique values in each column, just to find the categorical columns.
# Generally it is given in the description of data which columns are␣
 ↪categorical and which are continous.
for i in df.columns:
    print(i,df[i].nunique())
```

5.2 Encoding Categorical Labels

Label Encoder is used for converting categorical string columns to numeric. - Algorithms from sklearn do not accept input columns with string type, convert those columns to numerical.

- So, we need to convert such columns (e.g. "checking_balance" or "purpose" in this dataset) into numbers.

[ ]:
```
# Following coloumns are to be converted into srting
categorical_cols = ['checking_balance','credit_history',
                    'purpose','savings_balance','employment_length',
                    'personal_status','other_debtors','property',
                    'installment_plan','housing', 'job', 'telephone',
                    'foreign_worker']
```

[ ]:
```

```

[ ]:
```

```

[ ]:
```
# LabelEncoder is used for converting categorical string columns to numeric.
# Read more about LabelEncoder in sklearn documentation.

# Define Label Encoder Model (Create object of LabelEncoder Class)
le = LabelEncoder()
```

[ ]:
```
for col in categorical_cols:
    # Taking a column from dataframe, encoding it and replacing same column in␣
 ↪the dataframe.
    df[col] = le.fit_transform(df[col])
```

[ ]:
```
df.head()       # now all the string columns are converted into numbers
```

5.3 Split the data into train and test

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

[ ]:
```
# Split 80% train and 20 % test
x_train, x_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=5)
```

6 Training the model (Decison Tree)

[ ]:
```
# Creating object of the DT with required options
clf = DecisionTreeClassifier(criterion='entropy')
```

[ ]:
```
# Training/Build the model with train data
clf.fit(x_train,y_train)
```

[ ]:
```
clf.predict(x_test)
```

[ ]:
```
from matplotlib import pyplot as plt
```

[ ]:
```
fig = plt.figure(figsize = (20,20), dpi=600)
plot_tree(clf,feature_names = x_train.columns,
#                 class_names=y_train.unique(),
                filled = True);
fig.savefig("tree2.png")
```

[ ]:
```

```

[ ]:
```
# Make predictions on test data
predictions = clf.predict(x_test)
```

[ ]:
```
predictions
```

7 5. Evaluate the model

7.1 Confusion Matrix

[ ]:
```
confusion_matrix(y_test, predictions)
```

7.1.1 Display Confusion Matrix

[ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay
```

[ ]:
```
cm = confusion_matrix(y_test, predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=clf.classes_)
```

[ ]:
```
disp.plot()
```

7.2 Accuracy

[ ]:
```
accuracy_score(y_test,predictions)*100
```

7.3 Precision, Recall and F1-Score

Click here for image.

[ ]:
```
from sklearn.metrics import precision_score, recall_score, f1_score
```

[ ]:
```
# Precision (P)
precision_score(y_test,predictions)
```

[ ]:
```
# Recall (R)
recall_score(y_test,predictions)
```

[ ]:
```
# F1-Score
f1_score(y_test,predictions)
```