15th April Assignment

April 29, 2023

1 Assignment 70

Q1. You are working on a machine learning project where you have a dataset containing numerical and categorical features. You have identified that some of the features are highly correlated and there are missing values in some of the columns. You want to build a pipeline that automates the feature engineering process and handles the missing values?

Design a pipeline that includes the following steps:

- Use an automated feature selection method to identify the important features in the dataset. ### Create a numerical pipeline that includes the following steps.
- Impute the missing values in the numerical columns using the mean of the column values.
- Scale the numerical columns us#ng standardisation.

Create a categorical pipeline that includes the following steps

- Impute the missing values in the categorical columns using the most frequent value of the column.
- One-hot encode the categorical columns.
- Combine the numerical and categorical pipelines using a ColumnTransformer.
- Use a Random Forest Classifier to build the final model.
- Evaluate the accuracy of the model on the test dataset.

Note: Your solution should include code snippets for each step of the pipeline, and a brief explanation of each step. You should also provide an interpretation of the results and suggest possible improvements for the pipeline.

Q2. Build a pipeline that includes a random forest classifier and a logistic regression classifier, and then use a voting classifier to combine their predictions. Train the pipeline on the iris dataset and evaluate its accuracy.

Q2 solution

[1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
from sklearn.datasets import load_iris
import warnings
warnings.simplefilter('ignore')
```

[2]:
```
dataset = load_iris()
```

[3]:
```
df = pd.DataFrame(data=dataset.data,columns=dataset.feature_names)
df['Target'] = dataset.target
df.head()
```

[3]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   Target
0       0
1       0
2       0
3       0
4       0
```

[4]:
```
# segregate the feature into indepedent and dependent feature
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

[5]:
```
# train test and split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
 ↪20,random_state=42)
```

[6]:
```
# import logistic regression
from sklearn.linear_model import LogisticRegression
```

[8]:
```
lr = LogisticRegression()
```

[9]:
```
# import random forest classifier
from sklearn.ensemble import RandomForestClassifier
```

[10]:
```
rf = RandomForestClassifier()
```

[11]:
```
# import voting classifier
from sklearn.ensemble import VotingClassifier
```

[13]:
```
vc = VotingClassifier(estimators=[('rf',rf),('lr',lr)],voting='hard')
```

[14]:
```
# creating pipeline
from sklearn.pipeline import Pipeline
```

[15]:
```
pipeline = Pipeline([('classifier',vc)])
```

[16]:
```
# train the pipeline
pipeline.fit_transform(X_train,y_train)
```

[16]:
```
array([[0, 0],
       [0, 0],
       [1, 1],
       [0, 0],
       [0, 0],
       [2, 2],
       [1, 1],
       [0, 0],
       [0, 0],
       [0, 0],
       [2, 2],
       [1, 1],
       [1, 1],
       [0, 0],
       [0, 0],
       [1, 1],
       [2, 2],
       [2, 2],
       [1, 1],
       [2, 2],
       [1, 1],
       [2, 2],
       [1, 1],
       [0, 0],
       [2, 2],
       [1, 1],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 1],
       [2, 2],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 1],
       [0, 0],
       [1, 1],
       [2, 2],
       [0, 0],
[1, 1],
[2, 2],
[0, 0],
[2, 2],
[2, 2],
[1, 1],
[1, 1],
[2, 2],
[1, 1],
[0, 0],
[1, 1],
[2, 2],
[0, 0],
[0, 0],
[1, 1],
[1, 2],
[0, 0],
[2, 2],
[0, 0],
[0, 0],
[1, 2],
[1, 1],
[2, 2],
[1, 2],
[2, 2],
[2, 2],
[1, 1],
[0, 0],
[0, 0],
[2, 2],
[2, 2],
[0, 0],
[0, 0],
[0, 0],
[1, 1],
[2, 2],
[0, 0],
[2, 2],
[2, 2],
[0, 0],
[1, 1],
[1, 1],
[2, 2],
[1, 1],
[2, 2],
[0, 0],
[2, 2],
       [1, 1],
       [2, 2],
       [1, 1],
       [1, 1],
       [1, 1],
       [0, 0],
       [1, 1],
       [1, 1],
       [0, 0],
       [1, 1],
       [2, 2],
       [2, 2],
       [0, 0],
       [1, 1],
       [2, 2],
       [2, 2],
       [0, 0],
       [2, 2],
       [0, 0],
       [1, 1],
       [2, 2],
       [2, 2],
       [1, 1],
       [2, 2],
       [1, 1],
       [1, 1],
       [2, 2],
       [2, 2],
       [0, 0],
       [1, 1],
       [2, 2],
       [0, 0],
       [1, 1],
       [2, 2]])
```

[17]:

```
y_pred = pipeline.predict(X_test)
```

[18]:

```
# Metric Evaluation
from sklearn.metrics import accuracy_score
print('Accuracy score:',accuracy_score(y_test,y_pred))
```

```
Accuracy score: 1.0
```

Q1 solution

[19]:

```
# load tip dataset
df = sns.load_dataset('tips')
# read the data
```

```
df.head()
```

```
[19]:    total_bill   tip     sex smoker  day    time  size
      0       16.99  1.01  Female     No  Sun  Dinner     2
      1       10.34  1.66    Male     No  Sun  Dinner     3
      2       21.01  3.50    Male     No  Sun  Dinner     3
      3       23.68  3.31    Male     No  Sun  Dinner     2
      4       24.59  3.61  Female     No  Sun  Dinner     4
```

```
[20]: # information about dataset
      df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   total_bill  244 non-null    float64
 1   tip         244 non-null    float64
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
```

```
[27]: # view categorical features
      df.drop(labels=['time'],axis=1)
      categorical_cols = [col for col in df.columns if df[col].dtypes == 'category']
      categorical_cols.remove('time')
      categorical_cols
```

```
[27]: ['sex', 'smoker', 'day']
```

```
[28]: # view numerical features
      numerical_cols = [col for col in df.columns if df[col].dtypes != 'category']
      numerical_cols
```

```
[28]: ['total_bill', 'tip', 'size']
```

```
[29]: # check for missing values in whole dataset
      df.isnull().sum()
```

```
[29]: total_bill    0
      tip           0
      sex           0
```

```
smoker        0
day           0
time          0
size          0
dtype: int64
```

[30]:

```
# label encoding on categorical features
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df['time'] = encoder.fit_transform(df['time'])
```

[31]:

```
# segregate the features into independent and dependent
X = df.drop(labels=['time'],axis=1)
y = df['time']
```

[32]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,random_state=42)
```

[33]:

```
# importing dependencies for pipeline and feature engineering
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
categorical_cols = ['sex','smoker','day']
numerical_cols = ['total_bill','tip','size']
```

[34]:

```
numerical_cols = ['total_bill','tip','size']
# pipeline for feature engineering for numerical and categorical
num_pipeline = Pipeline(
    steps=[
        ('imputer',SimpleImputer(strategy='median')),
        ('scaler',StandardScaler())
    ]
)

cat_pipeline = Pipeline(
    steps=[
        ('imputer',SimpleImputer(strategy='most_frequent')),
        ('onehotencoder',OneHotEncoder())
    ]
)
```

[35]:

```
preprocessor = ColumnTransformer(
    [
        ('num_pipeline',num_pipeline,numerical_cols),
        ('cat_pipeline',cat_pipeline,categorical_cols)
    ]
)
```

[36]:

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

[37]:

```
# model training automating
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

[38]:

```
models = {
    'Random Forest': RandomForestClassifier(),
    'Decision Tress': DecisionTreeClassifier(),
    'Logistic Regressor': LogisticRegression(),
    'svc' : SVC()
}
```

[39]:

```
# performance evaluation
from sklearn.metrics import accuracy_score
```

[40]:

```
def evaluate_model(X_train,y_train,X_test,y_test,models):
    reports = {}
    for i in range(len(models)):
        model = list(models.values())[i]
        # train model
        model.fit(X_train,y_train)
        # predicting test data
        y_pred = model.predict(X_test)
        # Check for score
        score = accuracy_score(y_test,y_pred)
        reports[list(models.keys())[i]] = score
    return reports
```

[41]:

```
evaluate_model(X_train,y_train,X_test,y_test,models)
```

[41]:

```
{'Random Forest': 0.9591836734693877,
 'Decision Tress': 0.9387755102040817,
 'Logistic Regressor': 1.0,
 'svc': 0.9591836734693877}
```