ufrwjc0r4

September 13, 2023

[ ]:
```
# 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 using 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.
```

[11]:
```
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectFromModel
```

```python
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Load Your Dataset
data = pd.read_csv('Insurance_data.csv')

# Separate Target Variable
X = data.drop('Charges', axis=1)
y = data['Charges']

# Define Categorical Columns
categorical_columns = ['Sex', 'Smoker', 'Region']

# Split Data Into Train And Test Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Automated Feature Selection with One-Hot Encoding
feature_selector = SelectFromModel(RandomForestRegressor(n_estimators=100,␣
↪random_state=42))

# Define Numerical And Categorical Features
numerical_features = X_train.select_dtypes(include=['float64', 'int64']).
↪columns.tolist()
categorical_features = X_train.select_dtypes(include=['object']).columns.
↪tolist()

# Numerical Pipeline
numerical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

# Categorical Pipeline
categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

# Combine The Numerical And Categorical Pipelines Using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_pipeline, numerical_features),
        ('cat', categorical_pipeline, categorical_features)
    ])
```

```python
# Final Pipeline With Preprocessing And Model
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('feature_selector', feature_selector),
    ('regressor', RandomForestRegressor(n_estimators=100, random_state=42))
])

# Fit The Pipeline On The Training Data
pipeline.fit(X_train, y_train)

# Make Predictions On The Test Data
y_pred = pipeline.predict(X_test)

# Evaluate The Model (Use Mean Squared Error For Regression Tasks)
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
```

```
Mean Squared Error: 13932964.31
```

[26]:

```python
import pandas as pd
import numpy as np
import random

# Set A Random Seed For Reproducibility
np.random.seed(42)

# Generate Random Data For The Dataset
n_samples = 100

# Placeholder List Of Names
names = ["John", "Jane", "Alice", "Bob", "Eva", "David", "Sophia", "Michael",␣
 ↪"Olivia", "William"]

data = {

    'Name': [random.choice(names) for _ in range(n_samples)],
    'Age': np.random.randint(18, 65, size=n_samples),
    'Sex': np.random.choice(['Male', 'Female'], size=n_samples),
    'Education': np.random.choice(['High School', 'Bachelor', 'Master', 'PhD']),
    'Salary': np.random.randint(20000, 80000, size=n_samples),
    'Credit_Score': np.random.randint(300, 850, size=n_samples),
    'Loan_Grant': np.random.choice(['Yes', 'No'], size=n_samples)
}

# Create A Dataframe From The Random Data
df = pd.DataFrame(data)
```

```python
# Save The Dataframe To A CSV File
df.to_csv('Loan_Data.csv', index=False)
```

[27]:
```python
import pandas as pd
import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder

# Load The Dataset
data = pd.read_csv('Loan_Data.csv')

# Separate Target Variable
X = data.drop('Loan_Grant', axis=1)
y = data['Loan_Grant']

# Drop 'Name' Feature
X = X.drop('Name', axis=1)

# Define Categorical And Numerical Features
categorical_features = ['Sex', 'Education']
numerical_features = ['Age', 'Salary', 'Credit_Score']

# Split The Data Into Train And Test Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[28]:
```python
# Numerical Pipeline
numerical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

# Categorical Pipeline
categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])
```

[22]:

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_pipeline, numerical_features),
        ('cat', categorical_pipeline, categorical_features)
    ])
```

[29]:

```
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('feature_selector',␣
 ↪SelectFromModel(RandomForestClassifier(n_estimators=100, random_state=42))),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
])
```

[30]:

```
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
```

[31]:

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.35
```

1 The SelectFromModel IS A Feature Selection Technique In Scikit-learn That Allows You To Select The Most Important Features From A Dataset Based On The Importance Scores Assigned To Them By A Machine Learning Model. In This Case, I Am Using SelectFromModel With A RandomForestClassifier As The Model To Select Important Features For Classification.

2 N Estimators = 100: This Specifies The Number Of Decision Trees In The Forest. You Are Using 100 Decision Trees.

[32]:

```
# 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.
```

[33]:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import accuracy_score

# Load The Iris Dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split The Data Into Training And Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Create Individual Classifiers
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
lr_classifier = LogisticRegression(max_iter=1000, random_state=42)

# Create A Voting Classifier
voting_classifier = VotingClassifier(
    estimators=[
        ('random_forest', rf_classifier),
        ('logistic_regression', lr_classifier)
    ],
    voting='hard'  # You Can Use 'Soft' For Weighted Voting If Both Classifiers␣
↪Support Probability Estimates
)


# Create A Pipeline That Includes Scaling And The Voting Classifier
pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Standardize Features For Logistic␣
↪Regression
    ('voting_classifier', voting_classifier)
])

# Fit The Pipeline On The Training Data
pipeline.fit(X_train, y_train)


# Make Predictions On The Test Data
y_pred = pipeline.predict(X_test)


# Evaluate The Accuracy Of The Ensemble Model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 1.00
```

3 A Voting Classifier Is An Ensemble Machine Learning Model That Combines The Predictions Of Multiple Individual Classifiers (Estimators) To Make A Final Decision. It Can Be Used For Both Classification And Regression Tasks.