code

September 3, 2023

[20]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.impute import SimpleImputer

df = pd.read_csv('../../dataset/cross-validation.csv')

# Split the dataset into features (X) and the target variable (y)
y = (df['Loan_Status'] == 'Y').astype(int)
X = df.drop('Loan_Status', axis=1)

# Encode categorical variables using one-hot encoding
categorical_columns = ['Loan_ID', 'Gender', 'Married', 'Dependents',␣
↪'Education', 'Self_Employed', 'Property_Area']
encoder = OneHotEncoder(sparse_output=False, drop='first')  # Set␣
↪sparse_output=False
X_encoded = encoder.fit_transform(X[categorical_columns])

# Get the one-hot encoded feature names
encoded_feature_names = encoder.
↪get_feature_names_out(input_features=categorical_columns)

X_encoded_df = pd.DataFrame(X_encoded, columns=encoded_feature_names)
X = X.drop(categorical_columns, axis=1)
X = pd.concat([X, X_encoded_df], axis=1)

# Handle missing values using SimpleImputer (replace NaN with the mean of each␣
↪column)
imputer = SimpleImputer(strategy='mean')  # You can choose a different strategy␣
↪if needed
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Split the dataset into 80% training and 20% testing
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Scale the data using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train a Logistic Regression model with Saga solver and no regularization
model = LogisticRegression(solver='saga', penalty=None, max_iter=10000)  # Use␣
↪penalty=None

# Manually perform k-fold cross-validation and calculate evaluation measures
k = 5  # Number of folds
fold_size = len(X_train_scaled) // k
accuracy_scores = []
precision_scores = []
recall_scores = []

for i in range(k):
    start = i * fold_size
    end = (i + 1) * fold_size if i < k - 1 else len(X_train_scaled)

    X_train_fold = np.concatenate([X_train_scaled[:start], X_train_scaled[end:
↪]])
    y_train_fold = np.concatenate([y_train[:start], y_train[end:]])

    model.fit(X_train_fold, y_train_fold)

    # Predict on the validation set
    y_pred = model.predict(X_train_scaled[start:end])

    # Calculate accuracy, precision, and recall for this fold
    accuracy = np.mean(y_pred == y_train[start:end])
    precision = np.sum((y_pred == 1) & (y_train[start:end] == 1)) / np.
↪sum(y_pred == 1)
    recall = np.sum((y_pred == 1) & (y_train[start:end] == 1)) / np.sum(y_train.
↪iloc[start:end] == 1)

    accuracy_scores.append(accuracy)
    precision_scores.append(precision)
    recall_scores.append(recall)

# Calculate mean accuracy, precision, and recall
mean_accuracy = np.mean(accuracy_scores)
mean_precision = np.mean(precision_scores)
mean_recall = np.mean(recall_scores)
```

```python
# Print the results
print("Mean Accuracy:", mean_accuracy)
print("Mean Precision:", mean_precision)
print("Mean Recall:", mean_recall)
```

```
Mean Accuracy: 0.7373118944547516
Mean Precision: 0.795191803599134
Mean Recall: 0.8389684680701743
```