EARTHQUAKE PREDICTION MODEL USING PYTHON

AI_PHASE2

NAME : SUBASH RAJ V

REG.NO: 610821205052

PHASE 2: INNOVATION

Considering advanced techniques such as hyperparameter tuning and feature engineering to improve the prediction model's performance

Improving the performance of an earthquake prediction model involves optimizing hyperparameters and performing feature engineering. The step-by-step explanation with relevant Python code snippets using libraries like scikit-learn and pandas given.

Import Libraries and Load Data

- Importing the necessary libraries and loading the dataset set downloaded from the dataset link: https://www.kaggle.com/datasets/usgs/earthquake-database to perform the techniques.
- The required libraries are imported as it is represented, such as pandas, sklearn libraries.

1. Importing Libraries:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
```

These import statements bring in the necessary libraries and modules for data manipulation, machine learning, and preprocessing.

2. Load Earthquake Data:

```
data = pd.read_csv('earthquake_data.csv')
```

This code loads the earthquake data from a CSV file ('earthquake_data.csv') into a pandas DataFrame called `data`.

3. Split Data into Features and Target Labels:

```
X = data.drop('Latitude', axis=1)
y = data['Longitude']
```

Here, we split the data into two parts:

- `X`: This DataFrame contains the features (input variables) for the model. We remove the 'target_label' column using `drop` to create `X`.

- `y`: This Series contains the target labels (the values you want to predict), which is the 'target_label' column from the original dataset.

4. Split 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)
```

This code uses `train_test_split` to divide the data into training and testing sets. The training set (`X_train` and `y_train`) will be used to train the model, while the testing set (`X_test` and `y_test`) will be used to evaluate the model's performance. The `test_size` parameter specifies that 20% of the data should be reserved for testing, and `random_state` ensures reproducibility.

5. Define Categorical and Numerical Columns:

```
categorical_cols = ['categorical_feature1', 'categorical_feature2']
numerical_cols = ['numerical_feature1', 'numerical_feature2']
```

Here specifying columns in the dataset are categorical and which are numerical. Replace these column names with the actual column names from dataset.

6. Preprocessing Pipelines:

```
numerical_transformer = Pipeline(steps=[
```

```
    ('imputer', SimpleImputer(strategy='mean'))
])

categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

These pipelines define how to preprocess numerical and categorical data.

- `numerical_transformer`: It imputes missing numerical values using the mean strategy. You can customize the imputation strategy as needed.

- `categorical_transformer`: It performs one-hot encoding on categorical data, handling unknown categories by ignoring them.

7. Column Transformer for Preprocessing:

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])
```

The `ColumnTransformer` bundles the preprocessing steps for both numerical and categorical data, applying the respective transformers to each feature type.

8. Random Forest Classifier:

```
rf_classifier = RandomForestClassifier(random_state=42)
```

This code creates an instance of the Random Forest Classifier with a specified random seed (`random_state=42`) for reproducibility.

9. Create the Full Pipeline:

```
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('model', rf_classifier)])
```

The full pipeline combines the preprocessing (`preprocessor`) and the machine learning model (`rf_classifier`) into a single workflow.

HYPERPARAMETER TUNING:

10. Define Hyperparameters to Search:

```
param_grid = {
    'model__n_estimators': [100, 200, 300],
    'model__max_depth': [None, 10, 20, 30],
    'model__min_samples_split': [2, 5, 10],
    'model__min_samples_leaf': [1, 2, 4]
}
```

This dictionary defines the hyperparameters and their corresponding values to search for using grid search.

11. Grid Search with Cross-Validation:

```
grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2, error_score='raise')
```

`GridSearchCV` is used to perform a grid search with 5-fold cross-validation to find the best hyperparameters. It utilizes the full pipeline (`pipeline`) and the hyperparameter grid (`param_grid`).

12. Fit the Model with the Best Hyperparameters:

```
grid_search.fit(X_train, y_train)
```

This code fits the model with the training data using the best hyperparameters found during grid search.

13. Get the Best Hyperparameters:

```
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)
```

After grid search, this prints out the best hyperparameters that were found during the search.

14. Use the Best Model for Prediction:

```
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

We use the best model (which includes the best hyperparameters) to make predictions on the testing data.

15. Evaluate the Model's Performance:

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Finally, the code calculates and prints the accuracy score, which is a measure of the model's performance on the test set.

FEATURE ENGINEERING:

1. Creating a New Feature - Earthquake Magnitude Squared:

```
X['magnitude_squared'] = X['magnitude'] ** 2
```

- In this step, a new feature named 'magnitude_squared' is created in the DataFrame `X`.

- It is calculated by squaring the values in the existing 'magnitude' feature. This can be a meaningful transformation if the relationship between the squared magnitude and the target variable is non-linear and can improve the model's performance.

2. Splitting the Updated 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)
```

- This code splits the updated DataFrame `X` and the target labels `y` into training and testing sets using the `train_test_split` function.

- The `test_size` parameter specifies that 20% of the data should be reserved for testing, and `random_state` ensures reproducibility by fixing the random seed.

3. Retraining the Model with Updated Features:

```
best_model.fit(X_train, y_train)
```

- The `fit` method is called on the best model (`best_model`) to train it using the updated training data (`X_train` and `y_train`).

- This step reuses the best model found during hyperparameter tuning, which is assumed to be stored in the `best_model` variable.

4. Making Predictions and Evaluating:

```
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

- Here, predictions are made on the test data (`X_test`) using the trained `best_model`.

- The accuracy of the model is calculated by comparing the predicted labels (`y_pred`) with the true labels (`y_test`) using the `accuracy_score` function from scikit-learn.

- The accuracy score measures the proportion of correctly predicted labels in the test set.

5. Printing the Accuracy with Feature Engineering:

```
print("Accuracy with Feature Engineering:", accuracy)
```

- Finally, the code prints the accuracy of the model on the test set after feature engineering.

- This allows you to assess whether the addition of the squared magnitude feature improved the model's predictive performance.

By following these steps, advanced techniques such as hyperparameter tuning and feature engineering to improve the prediction model's can be performed.