Earthquake Prediction Model using Python

Development Part1

Certainly! Let's continue the development of your Earthquake Prediction Model in Python. We'll cover further steps, including model training, evaluation, and potential improvements.

Part 1: Model Training

python

```
# Choose a machine learning algorithm (e.g., RandomForestClassifier)
from sklearn.ensemble import RandomForestClassifier

# Initialize the model
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train, y_train)
```

Part 2: Make Predictions and Evaluate

python

```
# Make predictions
predictions = model.predict(X_test)

# Evaluate the model performance
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Accuracy
accuracy = accuracy_score(y_test, predictions)
print(f'Accuracy: {accuracy}')

# Classification Report
class_report = classification_report(y_test, predictions)
print(f'Classification Report:\n{class_report}')
```

```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, predictions)
print(f'Confusion Matrix:\n{conf_matrix}')
```

Part 3: Model Improvement (Optional)

Consider improving your model based on its performance:

- **Feature Importance:**

Analyze feature importance to understand which features contribute the most.

python

```
# Example: Display feature importance
feature_importance = model.feature_importances_
print('Feature Importance:')
for feature, importance in zip(X.columns, feature_importance):
    print(f'{feature}: {importance}')
```

- **Hyperparameter Tuning:**

Fine-tune hyperparameters for better performance.

python

```
# Example: GridSearchCV for hyperparameter tuning
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    # Add other hyperparameters based on your model
}

grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=3)
grid_search.fit(X_train, y_train)

# Get the best parameters
```

```
best_params = grid_search.best_params_

print(f'Best Parameters: {best_params}')


# Use the best model

best_model = grid_search.best_estimator_
```

Next Steps:

- **Visualization:**

Visualize results and insights.

- **Documentation:**

Document your code, model details, and results for future reference.

- **Deployment (if applicable):**

If you plan to deploy your model, consider frameworks like Flask or FastAPI.

Feel free to adapt these steps based on your specific project requirements. If you have any questions or need more guidance, feel free to ask!