EARTHQUAKE PREDICTION MODEL USING PYTHON

TEAM MEMBER
311121205304 - MUTHEESWARAN G

Project : Earthquake prediction model using python

INTRODUCTION

Developing an earthquake prediction model using Python represents a formidable challenge in the realm of geoscience and data science. Earthquake prediction remains an intricate and elusive field, primarily due to the complex and dynamic nature of geological processes that underlie seismic activity. Nevertheless, Python, with its powerful libraries and tools for data analysis, machine learning, and visualization, can serve as a valuable platform for constructing early warning systems. These systems aim to monitor and analyze seismic data, historical earthquake records, and relevant geological information to provide timely alerts or insights into the likelihood of seismic events. While true earthquake prediction remains uncertain, such models contribute to our understanding of seismic activity and assist in mitigating potential risks to human life and infrastructure by offering valuable insights and alerts based on data-driven analysis.

INNOVATION

In the second phase of our project, we are taking a bold step into the realm of innovation, acknowledging that a truly successful earthquake prediction goes beyond merely providing responses; it must do so accurately, robustly, and intelligently. The primary objectives of this phase are as follows:

ENHANCING ACCURACY

Enhancing the accuracy of an earthquake prediction model is a challenging task due to the inherent complexity and uncertainty of seismic activity. However, you can take several steps to improve the accuracy of your earthquake prediction or early warning system:

- **Feature Engineering**:
    - Carefully select and engineer features that are highly relevant to seismic activity. This may include factors like historical earthquake

data, fault line information, geological characteristics, and seismic sensor readings.

- **Data Quality**:
  - Ensure that your data is clean, consistent, and free from outliers. Poor data quality can significantly affect the performance of your model.
- **Advanced Machine Learning Techniques**:
  - Experiment with more advanced machine learning algorithms, such as deep learning models (e.g., convolutional neural networks) or ensemble methods (e.g., XGBoost or LightGBM), to capture complex patterns in the data.
- **Hyperparameter Tuning**:
  - Optimize the hyperparameters of your model using techniques like grid search or random search to find the best configuration for your specific problem.
- **Cross-Validation**:
  - Implement cross-validation techniques to ensure that your model's performance is consistent across different subsets of your data. This helps prevent overfitting.
- **Feature Selection**:
  - Use feature selection methods to identify the most relevant features for earthquake prediction. Removing irrelevant or redundant features can improve model performance.
- **Ensemble Models**:
  - Combine multiple models or predictions through ensemble techniques like stacking or bagging. This can help improve prediction accuracy.

IMPROVING ROBUSTNESS

Improving the robustness of an earthquake prediction model is essential to ensure its reliability and effectiveness in real-world applications. Robustness refers to the model's ability to perform consistently and accurately under various conditions and in the presence of uncertainties. Here are some strategies to enhance the robustness of your earthquake prediction model using Python:

- **Data Quality and Preprocessing**:
    - Ensure that your data preprocessing pipeline is robust by handling missing data, outliers, and data anomalies effectively.
    - Implement data augmentation techniques to generate synthetic data points that can improve the model's generalization.
- **Feature Engineering**:
    - Create informative features that are robust to variations in data. These features should capture the most relevant aspects of seismic activity.
    - Consider using domain-specific knowledge to engineer features that are resistant to noise.
- **Regularization Techniques**:
    - Apply regularization techniques like L1 and L2 regularization to prevent overfitting and make the model more robust to noise in the training data.
- **Cross-Validation**:
    - Implement robust cross-validation strategies, such as k-fold cross-validation, to assess the model's performance across different subsets of the data.
- **Ensemble Models**:
    - Create ensemble models by combining the predictions of multiple base models. Ensemble methods like bagging and boosting can improve robustness by reducing the impact of individual model errors.
- **Model Interpretability**:
    - Use interpretable models or techniques (e.g., decision trees) to gain insights into how the model makes predictions. This can help identify and address potential sources of error.

ENSEMBLE METHODS

Ensemble methods combine the predictions of multiple machine learning models to improve accuracy and generalization. By integrating these techniques, we aim to boost the earthquake predictive model.

#Ensemble_Tech

```
import pandas as pd

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier, AdaBoostClassifier

from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score, classification_report

from google.colab import files

uploaded=files.upload()

data = pd.read_csv('database.csv')

data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]

data.head()
```

	Date	Time	Latitude	Longitude	Depth	Magnitude
0	01/02/1965	13:44:18	19.246	145.616	131.6	6.0
1	01/04/1965	11:29:49	1.863	127.352	80.0	5.8
2	01/05/1965	18:05:58	-20.579	-173.972	20.0	6.2
3	01/08/1965	18:49:43	-59.076	-23.557	15.0	5.8
4	01/09/1965	13:32:50	11.938	126.427	15.0	5.8

```
import datetime
import time

timestamp = []
for d, t in zip(data['Date'], data['Time']):
```

```
    try:
        ts = datetime.datetime.strptime(d+' '+t, '%m/%d/%Y %H:%M:%S')
        timestamp.append(time.mktime(ts.timetuple()))
    except ValueError:
        # print('ValueError')
        timestamp.append('ValueError')
timeStamp = pd.Series(timestamp)
data['Timestamp'] = timeStamp.values
final_data = data.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp != 'ValueError']
final_data.head()
```

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	19.246	145.616	131.6	6.0	-157630542.0
1	1.863	127.352	80.0	5.8	-157465811.0
2	-20.579	-173.972	20.0	6.2	-157355642.0
3	-59.076	-23.557	15.0	5.8	-157093817.0
4	11.938	126.427	15.0	5.8	-157026430.0

```
X = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(18727, 3) (4682, 3) (18727, 2) (4682, 2)
```

```
from sklearn.ensemble import RandomForestRegressor

reg = RandomForestRegressor(random_state=42)
reg.fit(X_train, y_train)
reg.predict(X_test)
```

```
array([[  5.865,  42.024],
       [  5.826,  33.09 ],
       [  6.082,  39.741],
       ...,
       [  6.306,  23.059],
       [  5.96 , 592.283],
       [  5.808,  38.222]])
```

reg.score(X_test, y_test)

```
0.3926671400442392
```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
final_data = data.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp != 'ValueError']
final_data.head()

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	19.246	145.616	131.6	6.0	-157630542.0
1	1.863	127.352	80.0	5.8	-157465811.0
2	-20.579	-173.972	20.0	6.2	-157355642.0
3	-59.076	-23.557	15.0	5.8	-157093817.0
4	11.938	126.427	15.0	5.8	-157026430.0

X = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

```
(18727, 3) (4682, 3) (18727, 2) (4682, 2)
```

dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train) # Make predictions on the test data dt_predictions = dt_classifier.predict(X_test) # Evaluate the model's performance accuracy =

accuracy_score(y_test, dt_predictions) print(f"Decision Tree Accuracy: {accuracy:.2f}") # Print a classification report for more detailed evaluation
print(classification_report(y_test, dt_predictions))

```
Decision Tree Accuracy: 0.92
```

	precision	recall	f1-score	support
0	0.93	0.92	0.92	1942
1	0.9	0.91	0.91	1617

accuracy			0.92	3559
macro avg	0.92	0.92	0.92	3559
weighted avg	0.92	0.92	0.92	3559

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score

decision_tree = DecisionTreeClassifier(random_state=42)
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
gradient_boosting = GradientBoostingClassifier(n_estimators=100,
random_state=42)

decision_tree.fit(X_train, y_train)
random_forest.fit(X_train, y_train)
gradient_boosting.fit(X_train, y_train)

dt_predictions = decision_tree.predict(X_test)
rf_predictions = random_forest.predict(X_test)
gb_predictions = gradient_boosting.predict(X_test)

stacked_X = pd.DataFrame({'DecisionTree': dt_predictions, 'RandomForest':
rf_predictions, 'GradientBoosting': gb_predictions})
meta_classifier = DecisionTreeClassifier(random_state=42)
meta_classifier.fit(stacked_X, y_test)
stacked_predictions = meta_classifier.predict(stacked_X)
```

```
stacked_accuracy = accuracy_score(y_test, stacked_predictions)
print(f"Stacked Model Accuracy: {stacked_accuracy:.2f}")
```

```
Stacked Model Accuracy: 0.94
```

DEEP LEARNING ARCHITECTURE

NEURAL NETWORKS:

```
from keras.models import Sequential
from keras.layers import Dense

def create_model(neurons, activation, optimizer, loss):
model = Sequential()
model.add(Dense(neurons, activation=activation, input_shape=(3,)))
model.add(Dense(neurons, activation=activation))
model.add(Dense(2, activation='softmax'))
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
return model

from keras.wrappers.scikit_learn import KerasClassifier

model = KerasClassifier(build_fn=create_model, verbose=0)

# neurons = [16, 64, 128, 256]
neurons = [16]
# batch_size = [10, 20, 50, 100]
batch_size = [10]
epochs = [10]
# activation = ['relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear', 'exponential']
activation = ['sigmoid', 'relu']
# optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax',
'Nadam']
optimizer = ['SGD', 'Adadelta']
loss = ['squared_hinge']
```

```
param_grid = dict(neurons=neurons, batch_size=batch_size, epochs=epochs,
activation=activation, optimizer=optimizer, loss=loss)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(X_train, y_train)


print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.666684 using {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'loss': 'squar
ed_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.666684 (0.471398) with: {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'loss':
'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.000000 (0.000000) with: {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'loss':
'squared_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
0.666684 (0.471398) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squ
ared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.000000 (0.000000) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squ
ared_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
```

```
model = Sequential()
model.add(Dense(16, activation='relu', input_shape=(3,)))
model.add(Dense(16, activation='relu'))
model.add(Dense(2, activation='softmax'))

model.compile(optimizer='SGD', loss='squared_hinge', metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=10, epochs=20, verbose=1,
validation_data=(X_test, y_test))
```

```
Train on 18727 samples, validate on 4682 samples
Epoch 1/20
18727/18727 [==============================] - 4s 233us/step - loss: 0.5038 - acc: 0.9182 - v
al_loss: 0.5038 - val_acc: 0.9242
Epoch 2/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - v
al_loss: 0.5038 - val_acc: 0.9242
Epoch 3/20
18727/18727 [==============================] - 4s 228us/step - loss: 0.5038 - acc: 0.9182 - v
al_loss: 0.5038 - val_acc: 0.9242
Epoch 4/20
18727/18727 [==============================] - 4s 222us/step - loss: 0.5038 - acc: 0.9182 - v
al_loss: 0.5038 - val_acc: 0.9242
Epoch 5/20
18727/18727 [==============================] - 5s 262us/step - loss: 0.5038 - acc: 0.9182 - v
al_loss: 0.5038 - val_acc: 0.9242
Epoch 6/20
18727/18727 [==============================] - 4s 223us/step - loss: 0.5038 - acc: 0.9182 - v
al_loss: 0.5038 - val_acc: 0.9242
Epoch 7/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - v
al_loss: 0.5038 - val_acc: 0.9242
Epoch 8/20
```

CONCLUSION:

In conclusion, our project focused on earthquake prediction using various machine learning algorithms, including RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier, Decision Tree, and XGBoost. Through our analysis, we found that these algorithms have demonstrated promising results in predicting earthquakes. While the performance varied across models, all of them provided valuable insights into earthquake patterns and potential predictive factors.

The project highlights the potential of machine learning in enhancing earthquake prediction, which is a critical area of research with far-reaching implications for public safety. Our findings underscore the importance of data preprocessing and feature engineering in improving model performance.

Although our models have shown promise, it is essential to acknowledge that earthquake prediction is a complex and challenging task, and there are limitations in the dataset and model accuracy. Future work should aim to refine and expand on the existing models, explore additional features and data sources, and consider more advanced machine learning techniques.

Overall, this project represents a step toward leveraging machine learning for earthquake prediction, and it opens the door to further research and development in this vital field of study.