ARTIFICIAL INTELLIGENCE- GROUP 5

PROJECT: EARTHQUAKE PREDICTION MODEL USING PYTHON

PHASE 2: INNOVATION

MOHAMED SALI HESHAM:963321104035

Improving an earthquake prediction model involves several steps, including data preprocessing, feature engineering, model selection, hyperparameter tuning, and evaluation. Here's a step-by-step guide on how to use advanced techniques like hyperparameter tuning and feature engineering to enhance the performance of your earthquake prediction model using Python:

Step 1: **Data Preprocessing**

Data Collection: Obtain a dataset containing earthquake-related features such as location, depth, magnitude, etc. You can use resources like the USGS Earthquake Catalog.

Data Cleaning: Handle missing values, outliers, and noise in the dataset.

Splitting Data: Divide the dataset into training and testing sets. This allows you to evaluate the model's performance on unseen data.

Step 2: **Feature Engineering**

Feature engineering is crucial for improving model performance. It involves creating new features or modifying existing ones to better represent the underlying patterns in the data.

Domain Knowledge: Understand the domain and consult experts if necessary to identify relevant features.

Temporal Features: Extract time-related information like day of the week, hour, month, etc., which might have an impact on earthquakes.

Spatial Features: Incorporate location-based features such as distance to fault lines, tectonic plate boundaries, etc.

Statistical Features: Compute statistical measures like mean, median, standard deviation, etc., for various attributes.

Step 3: **Model Selection**

Choose an appropriate machine learning algorithm for your prediction task. For earthquake prediction, algorithms like Random Forest, Support Vector Machines, and Deep Learning models could be suitable candidates.

Step 4: **Hyperparameter Tuning**

Hyperparameters are settings for your machine learning model that are not learned from the data. Tuning them can significantly impact the model's performance.

Grid Search or Random Search: Use techniques like Grid Search or Random Search to find the best combination of hyperparameters. This involves specifying a range of values for each hyperparameter and letting the algorithm search through them.

Cross-Validation: Use techniques like k-fold cross-validation to evaluate the model's performance across different hyperparameter configurations.

Step 5: Model Evaluation

Evaluate the model's performance using appropriate metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or other domain-specific metrics.

Step 6: **Iterative Process**

Based on the evaluation results, iterate through steps 2 to 5, fine-tuning features and hyperparameters as necessary.

Step 7: **Deployment**

Once you're satisfied with the model's performance, deploy it in a real-world setting. This may involve creating an API, a web application, or integrating it into an existing system.

Here's how you can use hyperparameter tuning and feature engineering to improve an earthquake prediction model using Python:

Feature Engineering: Feature engineering is the process of using domain knowledge to create features that make machine learning algorithms work. If feature engineering is done correctly, it increases the predictive power of machine learning algorithms by creating features from raw data that help facilitate the machine learning process.

For an earthquake prediction model, you could create features like:

- **Lag Features**: You can shift the sequence of seismic signals in time and create additional features. This can help capture the temporal nature of the data.
- **Statistical Features**: You can calculate statistical properties of seismic signals like mean, median, standard deviation, etc.
- **Frequency Domain Features**: You can transform the data into the frequency domain (using FFT) and extract features from there.

How Feature Engineering works in the Earthquake Prediction?

- **Data Preparation**: Load and read the dataset, which includes features like date, time, latitude, longitude, depth, and magnitude.
- **Feature Creation**: Create new features that might be useful for the model. For example, you could create a feature that represents the "distance from the epicenter to the nearest tectonic plate boundary". This could be useful because earthquakes often occur near tectonic plate boundaries.
- **Feature Transformation**: Transform existing features to make them more useful for the model. For example, you might convert the date and time of each earthquake to Unix time (the number of seconds since 1970-01-01 00:00:00 UTC). This numerical representation might be easier for the model to work with than the raw date and time.
- **Feature Scaling**: Scale features so that they're all in a similar range. This can help prevent features with larger scales from dominating those with smaller scales. For example, you might use Min-Max scaling or Standard scaling.
- **Feature Selection**: Select the most useful features for your model. This can help reduce overfitting and improve computational efficiency. You might use techniques like mutual information, chi-squared test, or recursive feature elimination for this.

Hyperparameter Tuning: Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is set before the learning process begins.

You can use methods like Grid Search or Random Search with Cross Validation for hyperparameter tuning. In Python, you can use libraries like GridSearchCV or RandomizedSearchCV from sklearn.model_selection.

Here's a simple example of how you might use Grid Search for hyperparameter tuning on a RandomForest model:

```
Python

 from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import GridSearchCV

# Define the model

model = RandomForestRegressor()

# Define the parameters to tune

param_grid = {

    'n_estimators': [50, 100, 200],

    'max_depth': [None, 10, 20, 30],

    'min_samples_split': [2, 5, 10]

}

# Perform grid search

grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
cv=3)

grid_search.fit(X_train, y_train)

# Get the best parameters

best_params = grid_search.best_params_
```

Earthquake Prediction Model using Laboratory Data

Forecasting earthquakes is one of the most important problems in Earth science because of their devastating consequences. Current scientific studies related to earthquake forecasting focus on three key points: when the event will occur, where it will occur, and how large it will be.

In this notebook we will work on addressing when the earthquake will take place. We will try to predict the time remaining before laboratory earthquakes occur from real-time seismic data.

Predicting earthquake can save billions of dollars in infrastructure but above all a large number of human lives can be saved.

```
import numpy as np
import pandas as pd
import os
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from catboost import CatBoostRegressor, Pool
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

#print(os.listdir("../input"))
```

1) Load the Data

Direct Link: Significant Earthquakes, 1965-2016 (kaggle.com)

```
train=pd.read_csv("../input/train.csv", nrows=6000000,dtype={'acoustic_
data':np.int16,'time_to_failure':np.float64})
train.head(10)
```

	acoustic_data	time_to_failure
0	12	1.4691
1	6	1.4691
2	8	1.4691
3	5	1.4691
4	8	1.4691
5	8	1.4691
6	9	1.4691
7	7	1.4691
8	-5	1.4691
9	3	1.4691

2) Exploratory Data Analysis

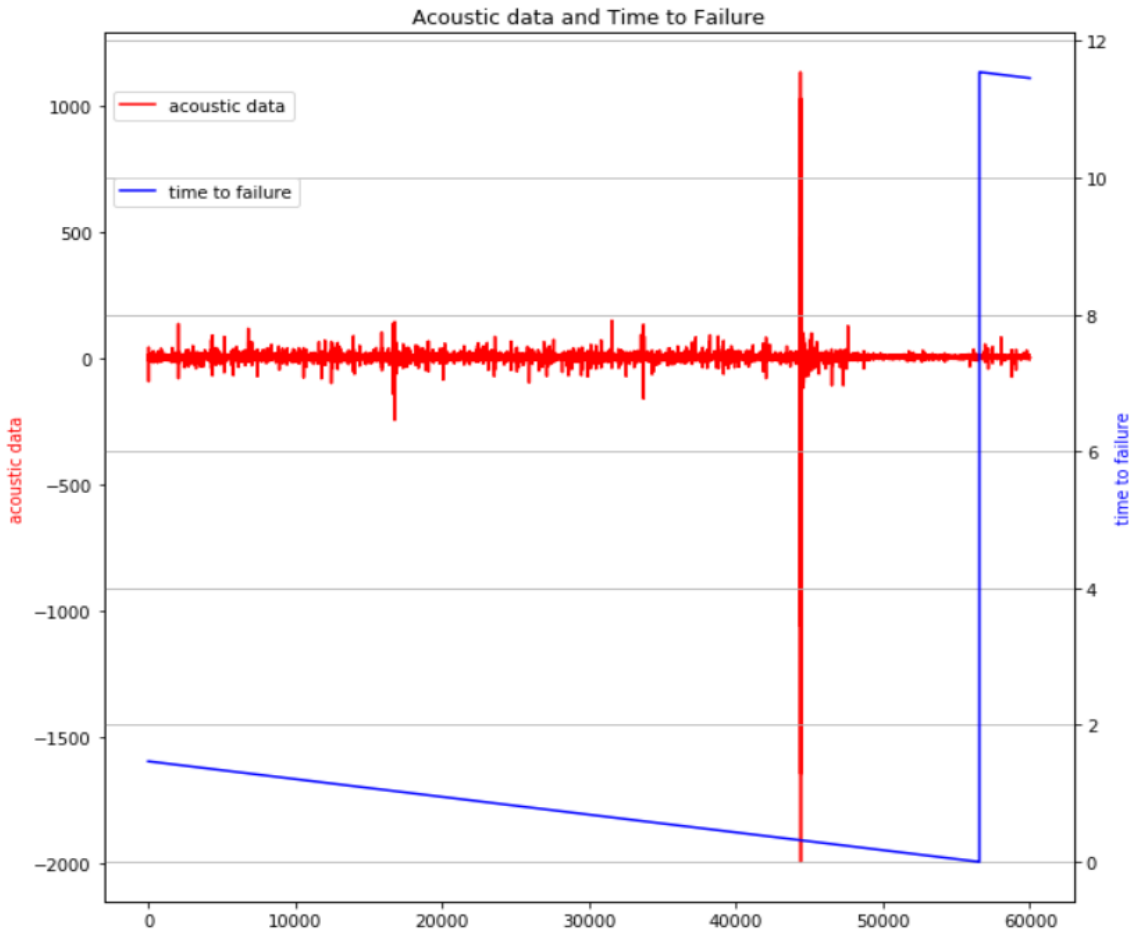
Lets plot the data to see and understand the data columns and our problem .
#We will use a small subset of dataset for understanding the pattern ,since the d
ata is large

```
train_acoustic_df = train['acoustic_data'].values[::100]
train_time_to_failure_df = train['time_to_failure'].values[::100]
```

```
fig, ax1 = plt.subplots(figsize=(10,10))
plt.title('Acoustic data and Time to Failure')
plt.plot(train_acoustic_df, color='r')
ax1.set_ylabel('acoustic data', color='r')
plt.legend(['acoustic data'], loc=(0.01, 0.9))

ax2 = ax1.twinx()
plt.plot(train_time_to_failure_df, color='b')
ax2.set_ylabel('time to failure', color='b')
plt.legend(['time to failure'], loc=(0.01, 0.8))

plt.grid(True)
```



The size of Train data is large . The two columns in the train dataset have the following meaning:

1. accoustic_data: is the accoustic signal measured in the laboratory experiment;
2. time to failure: this gives the time until a failure will occurs.

The above plot shows that the failure occur after a large oscilation and also that the large oscilation is followed by a series of small minor oscilations before the final time of failure .

3) Feature Engineering

```
def gen_features(X):
    fe = []
    fe.append(X.mean())
    fe.append(X.std())
    fe.append(X.min())
    fe.append(X.max())
    fe.append(X.kurtosis())
    fe.append(X.skew())
    fe.append(np.quantile(X,0.01))
    fe.append(np.quantile(X,0.05))
    fe.append(np.quantile(X,0.95))
    fe.append(np.quantile(X,0.99))
    fe.append(np.abs(X).max())
    fe.append(np.abs(X).mean())
    fe.append(np.abs(X).std())
    return pd.Series(fe)
```

```
#Lets read the training set again now in chunks and append features
train = pd.read_csv('../input/train.csv', iterator=True, chunksize=150_000, dtype
={'acoustic_data': np.int16, 'time_to_failure': np.float64})

X_train = pd.DataFrame()
y_train = pd.Series()
for df in train:
    ch = gen_features(df['acoustic_data'])
    X_train = X_train.append(ch, ignore_index=True)
    y_train = y_train.append(pd.Series(df['time_to_failure'].values[-1]))
```

```
X_train.head(10) #Let's check the training dataframe
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	4.884113	5.101106	-98.0	104.0	33.662481	-0.024061	-8.0	-2.0	11.0	18.0	104.0	5.576567	4.333325
1	4.725767	6.588824	-154.0	181.0	98.758517	0.390561	-11.0	-2.0	12.0	21.0	181.0	5.734167	5.732777
2	4.906393	6.967397	-106.0	140.0	33.555211	0.217391	-15.0	-3.0	13.0	26.0	140.0	6.152647	5.895945
3	4.902240	6.922305	-199.0	197.0	116.548172	0.757278	-12.0	-2.0	12.0	22.0	199.0	5.933960	6.061214
4	4.908720	7.301110	-126.0	145.0	52.977905	0.064531	-15.0	-2.0	12.0	26.0	145.0	6.110587	6.329485
5	4.913513	5.434111	-144.0	142.0	50.215147	-0.100697	-10.0	-2.0	12.0	19.0	144.0	5.695167	4.608383
6	4.855660	5.687823	-78.0	120.0	23.173004	0.208810	-12.0	-2.0	12.0	21.0	120.0	5.791007	4.732118
7	4.505427	5.854512	-134.0	139.0	52.388738	-0.176333	-11.0	-2.0	11.0	20.0	139.0	5.415000	5.025126
8	4.717833	7.789643	-156.0	168.0	65.360261	-0.160166	-16.0	-3.0	13.0	26.0	168.0	6.152273	6.714605
9	4.730960	6.890459	-126.0	152.0	53.760207	0.150779	-14.0	-3.0	12.0	23.0	152.0	5.925120	5.895191

```
submission = pd.read_csv('../input/sample_submission.csv', index_col='seg_id')
#Taking the segment id from sample_submission file

#Applying Feature Engineering on test data files
X_test = pd.DataFrame()
for seg_id in submission.index:
    seg = pd.read_csv('../input/test/' + seg_id + '.csv')
    ch = gen_features(seg['acoustic_data'])
    X_test = X_test.append(ch, ignore_index=True)

linkcode
X_test.head(10) #Lets check the testing dataframe
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	4.491780	4.893690	-75.0	115.0	28.837568	0.327908	-8.0	-2.0	11.0	18.0	115.0	5.224607	4.102161
1	4.171153	5.922839	-140.0	152.0	56.218955	0.295708	-12.0	-2.0	11.0	20.0	152.0	5.198340	5.045369
2	4.610260	6.946990	-193.0	248.0	162.118284	0.428688	-11.0	-2.0	11.0	20.0	248.0	5.597193	6.179525
3	4.531473	4.114147	-93.0	85.0	41.241827	0.061889	-5.0	-1.0	10.0	14.0	93.0	4.961487	3.583863
4	4.128340	5.797164	-147.0	177.0	79.539708	0.073898	-10.0	-2.0	10.0	19.0	177.0	5.070900	4.993617
5	4.148607	24.782769	-675.0	671.0	159.651642	1.016361	-52.0	-12.0	20.0	58.0	675.0	9.760847	23.154307
6	4.113987	4.707150	-107.0	125.0	57.044041	0.081861	-7.0	-2.0	10.0	15.0	125.0	4.785213	4.022915
7	4.328380	5.964443	-120.0	120.0	35.504290	0.161541	-13.0	-3.0	12.0	21.0	120.0	5.433700	4.978382
8	4.000733	5.874469	-114.0	118.0	37.260235	-0.007195	-12.0	-3.0	11.0	20.0	118.0	5.124560	4.924842
9	4.458800	8.926476	-258.0	281.0	133.899159	-0.123981	-17.0	-3.0	12.0	26.0	281.0	6.063680	7.924301

4) Catboost Regressor

```
#Catboost regressor model
"""
#Catboost Regressor model

train_pool = Pool(X_train, y_train)

m = CatBoostRegressor(iterations=10000, loss_function='MAE', boosting_type
='Ordered')
m.fit(X_train, y_train, silent=True)
m.best_score_
"""
```

```
"       \n#Catboost Regressor model\n\ntrain_pool = Pool(X_train, y_train)\n\nm
= CatBoostRegressor(iterations=10000, loss_function='MAE', boosting_type='
Ordered')\nm.fit(X_train, y_train, silent=True)\nm.best_score_\n"
```

5) Scale the Data

```
#Scale Train Data
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = pd.DataFrame(scaler.transform(X_train))
X_train_scaled.head(10)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1.424262	-0.170139	0.193132	-0.218112	-0.491116	-0.313682	0.228611	0.078754	-0.098277	-0.163239	-0.222489	0.019250	-0.169921
1	0.805767	0.004826	-0.018144	0.064045	0.431916	0.554010	0.015923	0.078754	0.325850	0.047674	0.036900	0.123150	-0.002085
2	1.511287	0.049349	0.162950	-0.086194	-0.492637	0.191611	-0.267660	-0.347453	0.749976	0.399197	-0.101216	0.399036	0.017483
3	1.495064	0.044046	-0.187919	0.122675	0.684165	1.321450	-0.054973	0.078754	0.325850	0.117979	0.097536	0.254865	0.037304
4	1.520375	0.088596	0.087494	-0.067873	-0.217232	-0.128283	-0.267660	0.078754	0.325850	0.399197	-0.084373	0.371307	0.069478
5	1.539098	-0.130976	0.019584	-0.078866	-0.256406	-0.474060	0.086819	0.078754	0.325850	-0.092935	-0.087742	0.097438	-0.136934
6	1.313125	-0.101138	0.268588	-0.159482	-0.639852	0.173654	-0.054973	0.078754	0.325850	0.047674	-0.168590	0.160622	-0.122094
7	-0.054873	-0.081534	0.057312	-0.089859	-0.225586	-0.632347	0.015923	0.078754	-0.098277	-0.022630	-0.104585	-0.087264	-0.086954
8	0.774779	0.146050	-0.025689	0.016408	-0.041656	-0.598514	-0.338556	-0.347453	0.749976	0.399197	-0.006893	0.398790	0.115665
9	0.826051	0.040300	0.087494	-0.042222	-0.206139	0.052211	-0.196765	-0.347453	0.325850	0.188283	-0.060792	0.249037	0.017393

```
#We will also scale the train data
X_test_scaled = pd.DataFrame(scaler.transform(X_test))
X_test_scaled.head(10)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	-0.108176	-0.194533	0.279906	-0.177804	-0.559531	0.422894	0.228611	0.078754	-0.098277	-0.163239	-0.185434	-0.212782	-0.197645
1	-1.360531	-0.073498	0.034675	-0.042222	-0.171275	0.355508	-0.054973	0.078754	-0.098277	-0.022630	-0.060792	-0.230099	-0.084526
2	0.354602	0.046949	-0.165282	0.309558	1.330328	0.633801	0.015923	0.078754	-0.098277	-0.022630	0.262602	0.032849	0.051493
3	0.046864	-0.286212	0.211996	-0.287735	-0.383644	-0.133811	0.441299	0.504961	-0.522404	-0.444458	-0.259545	-0.386246	-0.259804
4	-1.527759	-0.088278	0.008266	0.049387	0.159402	-0.108679	0.086819	0.078754	-0.522404	-0.092935	0.023425	-0.314114	-0.090733
5	-1.448598	2.144554	-1.983763	1.859589	1.295353	1.863642	-2.890812	-4.183314	3.718863	2.648943	1.701031	2.777771	2.087274
6	-1.583822	-0.216471	0.159177	-0.141160	-0.159576	-0.092015	0.299507	0.078754	-0.522404	-0.374153	-0.151747	-0.502456	-0.207149
7	-0.746410	-0.068605	0.110131	-0.159482	-0.465000	0.074733	-0.125869	-0.347453	0.325850	0.047674	-0.168590	-0.074936	-0.092560
8	-2.026185	-0.079187	0.132768	-0.166811	-0.440101	-0.278386	-0.054973	-0.347453	-0.098277	-0.022630	-0.175327	-0.278739	-0.098981
9	-0.236995	0.279749	-0.410513	0.430482	0.930194	-0.522788	-0.409452	-0.347453	0.325850	0.399197	0.373768	0.340384	0.260744

6) Support Vector Regression

We will be using the SVR model for prediction and GridSearchCV for hyperparameter tuning of the model

```
parameters = [{'gamma': [0.001, 0.005, 0.01, 0.02, 0.05, 0.1],
               'C': [0.1, 0.2, 0.25, 0.5, 1, 1.5, 2]}]

reg1 = GridSearchCV(SVR(kernel='rbf', tol=0.01), parameters, cv=5, scoring='
neg_mean_absolute_error')
```

```
reg1.fit(X_train_scaled, y_train.values.flatten())
```

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
  gamma='auto_deprecated', kernel='rbf', max_iter=-1, shrinking=True,
  tol=0.01, verbose=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid=[{'gamma': [0.001, 0.005, 0.01, 0.02, 0.05, 0.1], 'C': [0.1, 0.2, 0
.25, 0.5, 1, 1.5, 2]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='neg_mean_absolute_error', verbose=0)
```

```
submission.time_to_failure = reg1.predict(X_test_scaled)
submission
```

	time_to_failure
seg_id	
seg_00030f	5.012140
seg_0012b5	5.227187
seg_00184e	5.250233
seg_003339	7.423845
seg_0042cc	6.389104
seg_004314	4.157657
seg_004cd2	6.608829
seg_004ee5	3.109519
seg_004f1f	4.569246
seg_00648a	3.117575
seg_006e4a	2.992502
seg_007a37	4.529122
seg_00a37e	3.492720
seg_00be11	2.684836
seg_00c35b	7.701314
seg_00cc91	3.616717
seg_00e5f7	6.377334
seg_00f3b9	3.393147
seg_010eab	6.255316
seg_0125d9	5.131381
seg_0144cb	4.666594
seg_0165c6	8.725955
seg_016913	2.768775
seg_016cdb	4.842767
seg_017314	7.411448
seg_01a8dc	4.782327
seg_01c775	7.603821
seg_01ecb0	4.788653
seg_02042f	3.004076
seg_025e78	6.088580
...	...
seg_fcb7d0	8.768930
seg_fcca66	5.096814
seg_fcd32e	5.598638
seg_fd19b8	9.263416
seg_fd374e	6.375058
seg_fd6340	5.846372
seg_fd8add	6.281723

seg_fdba93	3.755962
seg_fdc1d4	3.575450
seg_fdd50e	2.737087
seg_fde767	4.836021
seg_fde86d	6.534552
seg_fdff11	1.772952
seg_fe2aac	3.089063
seg_fe5945	2.772504
seg_fe70c0	6.519753
seg_fe73b1	8.948574
seg_fe86f5	5.017798
seg_fea3e2	8.433415
seg_feb312	6.219464
seg_fedbd1	7.278150
seg_ff0f1b	3.259400
seg_ff1a62	7.443817
seg_ff2f2d	6.008494
seg_ff4236	4.306095
seg_ff7478	5.979068
seg_ff79d9	4.657432
seg_ffbd6a	2.112183
seg_ffe7cc	8.656368

Submission

```
submission.to_csv('submission.csv',index=True)
```