Deep Learning Programming Using Python Case Study: Earthquake Prediction System

ABSTRACT

Python programming language is reliable enough to solve Machine Learning and Deep Learning problems. This paper describes how to solve earthquake prediction problems using the Python programming language that runs in the Jupyter Notebook environment. With the python library used, namely Keras. Deep Learning programming for this earthquake prediction system is the following programming sequence: data preparation, Keras model determination, Keras model compilation, Keras model adjustment, Keras model evaluation, and prediction system creation. From the test results of the earthquake prediction system using the python programming language, the results are quite satisfying. The simulation results show the results of the Deep Learning training process for the prediction system of b-value as an earthquake precursor with several iterations of 10,000 times, the results of MSE, RMSE, MAPE, and the percentage of successful predictions are 5.43 x 10-5; 0.00737; 0.80897 and 99.19% respectively. The results of the Deep Learning testing process for the b-value prediction system as an earthquake precursor which was carried out during the five tests obtained an average of MSE, RMSE, MAPE and the percentage of successful predictions was 0.03886; 0.19003; 23.96459, and 77.75%.

Introduction

Python is a multipurpose interpretive programming language. Unlike some other languages which are relatively difficult to read and understand, python places more emphasis on code readability to make it easier to understand the syntax. This makes Python very easy to learn for both beginners and those who have mastered other programming languages. Some many modules and libraries can be used to implement Machine Learning and Deep Learning in Python. As in this paper, the Keras library is used for deep learning programming. And in particular, deep learning in this paper is tested to solve the b-value prediction system as an earthquake precursor.

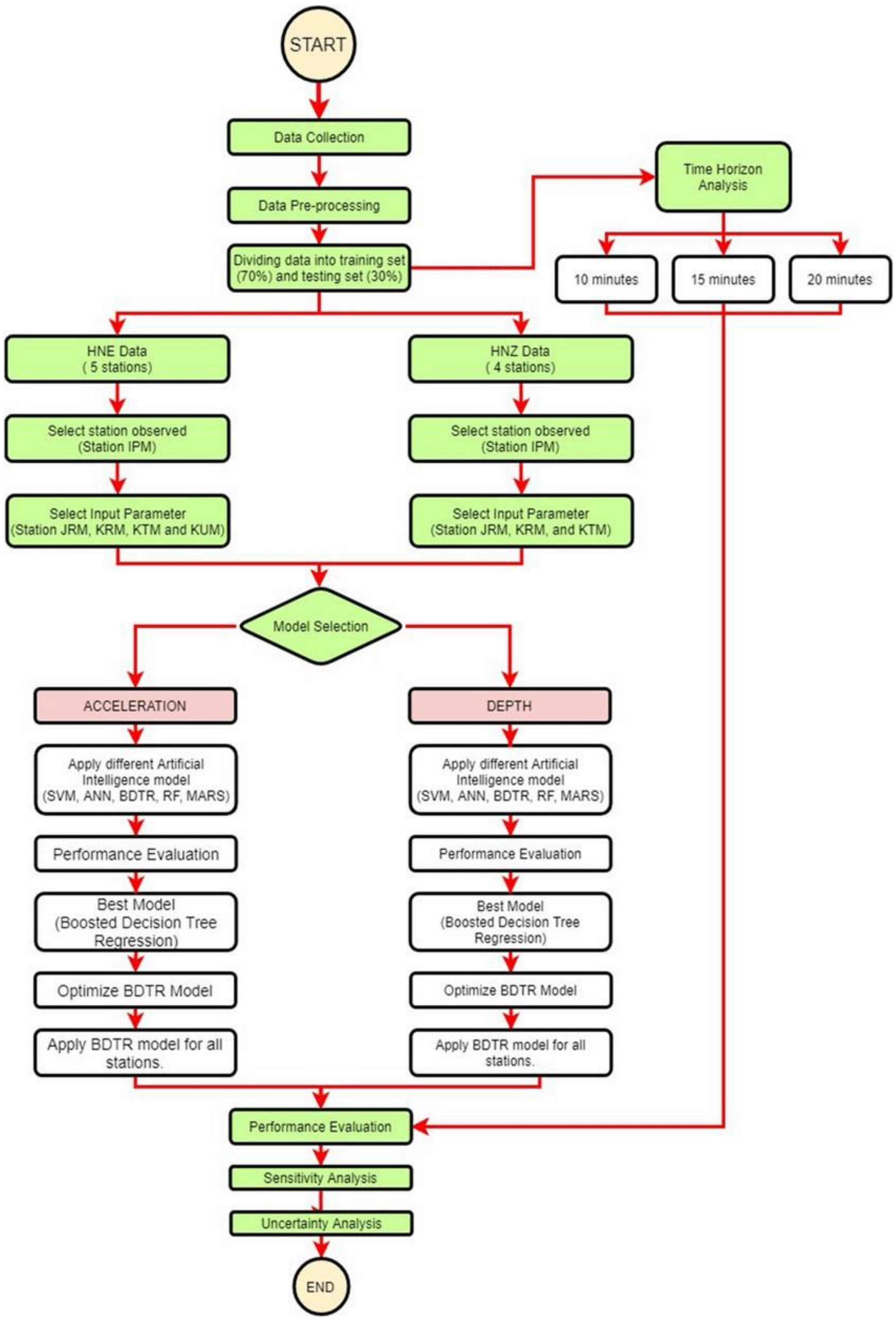
Nowadays machine learning is the branch of computer science that studies algorithm design that can be learned. Deep Learning is a sub-field of machine learning development which is a set of algorithms inspired by the structure and function of the brain. This algorithm is usually called an Artificial Neural Network (ANN). Deep learning is a development of ANN one of the hottest fields in data science with many case studies that have had amazing results in the fields of robotics, image recognition, and Artificial Intelligence (AI).

One of the most reliable and easy-to-use open-source Python libraries for developing and evaluating deep learning models in Keras; It wraps up the efficient numerical computation libraries Theano and TensorFlow. The advantage is that neural networks are easy to implement and fun. And it's possible to define and train a neural network model in just a few lines of code. Lots of deep learning programming applications use the Keras library, especially for prediction systems. Some of them can be mentioned among others: Deep learning-based prediction of species-specific protein Sglutathionylation sites (Li et al., 2020). Portfolio optimization with return prediction using deep learning and machine learning (Ma et al., 2021). The molecular structure incorporated a deep learning approach for accurate interfacial tension predictions (Yang et al., 2020). Prediction and analysis of COVID-19 positive cases using deep learning models: A descriptive case study of India (Arora et al., 2020). DeepPPSite: A deep learningbased model for analysis and prediction of phosphorylation sites using efficient sequence information (Ahmed et al., 2021). An end-to-end model for rice yield prediction using deep learning fusion (Chu & Yu, 2020). Tool wear mechanism and prediction in milling TC18 titanium alloy using deep learning (Ma et al., 2020). Predictions for COVID-19 with deep learning models of LSTM, GRU, and Bi-LSTM (Shahid et al., 2020). ETH analysis and predictions utilizing deep learning (Zoumpekas et al., 2020). Occupant-centric miscellaneous electric loads prediction in buildings using state-of-the-art deep learning methods (Das et al., 2020).

Material and Methods

As described in the introduction, this paper will show that python is effective enough to complete deep learning programming, with an example of an earthquake prediction system. The prediction referred to in this paper is a prediction of the b-value as an earthquake precursor. The data used in this study are earthquake data from the catalog of the International Seismological Center (ISC) Sumatra-Andaman region, which includes the boundaries of 92 ° 106 ° East Longitude (EL) and 6.5 ° South Latitude (SL) - 8 ° North Latitude (NL), the period January 1973 - November 2014. Magnitude greater than 3.0 SR, with a depth of less than 300 km (Rahmat et al., ____).

Under the dataset used, which consisted of a 444-month earthquake, 12 x 32 months, or 384 months were used for the training process. And the remaining 60 months are used for the validation process. Furthermore, from this data structure, each is designed for training flowcharts and deep learning testing flowcharts as in Figure 1 and Figure 2.


START
Data Collection
Data Pre-processing
Dividing data into training set (70%) and testing set (30%)
Time Horizon Analysis
10 minutes
15 minutes
20 minutes
HNE Data (5 stations)
HNZ Data (4 stations)
Select station observed (Station IPM)
Select station observed (Station IPM)
Select Input Parameter (Station JRM, KRM, KTM and KUM)
Select Input Parameter (Station JRM, KRM, and KTM)
Model Selection
ACCELERATION
DEPTH
Apply different Artificial Intelligence model (SVM, ANN, BDTR, RF, MARS)
Apply different Artificial Intelligence model (SVM, ANN, BDTR, RF, MARS)
Performance Evaluation
Performance Evaluation
Best Model (Boosted Decision Tree Regression)
Best Model (Boosted Decision Tree Regression)
Optimize BDTR Model
Optimize BDTR Model
Apply BDTR model for all stations.
Apply BDTR model for all stations.
Performance Evaluation
Sensitivity Analysis
Uncertainty Analysis
END

Earthquake Prediction Model with Machine Learning

In this article, I will take you through how to create a model for the task of Earthquake Prediction using Machine Learning and the Python programming language. Predicting earthquakes is one of the great unsolved problems in the earth sciences.

With the increase in the use of technology, many seismic monitoring stations have increased, so we can use machine learning and other data-driven methods to predict earthquakes.

Also, Read – Machine Learning Full Course for free.

Earthquake Prediction Model with Machine Learning
It is well known that if a disaster occurs in one region, it is likely to happen again. Some regions have frequent earthquakes, but this is only a comparative amount compared to other regions.

So, predicting the earthquake with date and time, latitude and longitude from previous data is not a trend that follows like other things, it happens naturally.

I will start this task to create a model for earthquake prediction by importing the necessary python libraries:

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Now let's load and read the dataset. The dataset that I am using here can be easily downloaded here:

```
data = pd.read_csv("database.csv")
data.columns
Index(['Date', 'Time', 'Latitude', 'Longitude', 'Type', 'Depth', 'Depth Error',
       'Depth Seismic Stations', 'Magnitude', 'Magnitude Type',
       'Magnitude Error', 'Magnitude Seismic Stations', 'Azimuthal Gap',
       'Horizontal Distance', 'Horizontal Error', 'Root Mean Square', 'ID',
       'Source', 'Location Source', 'Magnitude Source', 'Status'],
      dtype='object')
```

Now let's see the main characteristics of earthquake data and create an object of these characteristics, namely, date, time, latitude, longitude, depth, magnitude:

```
data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]
data.head()
```

Output:

```
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
```

Since the data is random, so we need to scale it based on the model inputs. In this, we convert the given date and time to Unix time which is in seconds and a number. This can be easily used as an entry for the network we have built:

```
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()
```

Output:

```
Latitude          Longitude        Depth  Magnitude       Timestamp
0         19.246 145.616          131.6  6.0       -1.57631e+08
1         1.863   127.352          80.0    5.8       -1.57466e+08
2         -20.579           -173.972        20.0    6.2       -1.57356e+08
3         -59.076           -23.557          15.0    5.8       -1.57094e+08
4         11.938 126.427          15.0    5.8       -1.57026e+08
```

Data Visualization

Now, before we create the earthquake prediction model, let’s visualize the data on a world map that shows a clear representation of where the earthquake frequency will be more:

```
from mpl_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urcrnrlon=180,lat_ts=20,resolution='c')

longitudes = data["Longitude"].tolist()
latitudes = data["Latitude"].tolist()
#m = Basemap(width=12000000,height=9000000,projection='lcc',
         #resolution=None,lat_1=80.,lat_2=55,lat_0=80,lon_0=-107.)
x,y = m(longitudes,latitudes)

fig = plt.figure(figsize=(12,10))
plt.title("All affected areas")
m.plot(x, y, "o", markersize = 2, color = 'blue')
m.drawcoastlines()
```

```
m.fillcontinents(color='coral',lake_color='aqua')
m.drawmapboundary()
m.drawcountries()
plt.show()
```

earthquake prediction

Splitting the Dataset

Now, to create the earthquake prediction model, we need to divide the data into Xs and ys which respectively will be entered into the model as inputs to receive the output from the model.

Here the inputs are TImestamp, Latitude and Longitude and the outputs are Magnitude and Depth. I’m going to split the xs and ys into train and test with validation. The training set contains 80% and the test set contains 20%:

```
X = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]
from sklearn.cross_validation 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, X_test.shape)
```

Ouput:

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

Neural Network for Earthquake Prediction

Now I will create a neural network to fit the data from the training set. Our neural network will consist of three dense layers each with 16, 16, 2 nodes and reread. Relu and softmax will be used as activation functions:

```
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
```

Now I’m going to define the hyperparameters with two or more options to find the best fit:

```
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)
```

Now we need to find the best fit of the above model and get the mean test score and standard deviation of the best fit model:

Program:

```
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))
```

Output:

```
Best: 0.957655 using {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'} 0.333316 (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'} 0.957655 (0.029957) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'} 0.645111 (0.456960) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
```

In the step below, the best-fit parameters are used for the same model to calculate the score with the training data and the test data:

Program:

```
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))

[test_loss, test_acc] = model.evaluate(X_test, y_test)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))
```

Output:

```
Evaluation result on Test Data : Loss = 0.5038455790406056, accuracy = 0.9241777017858995
```

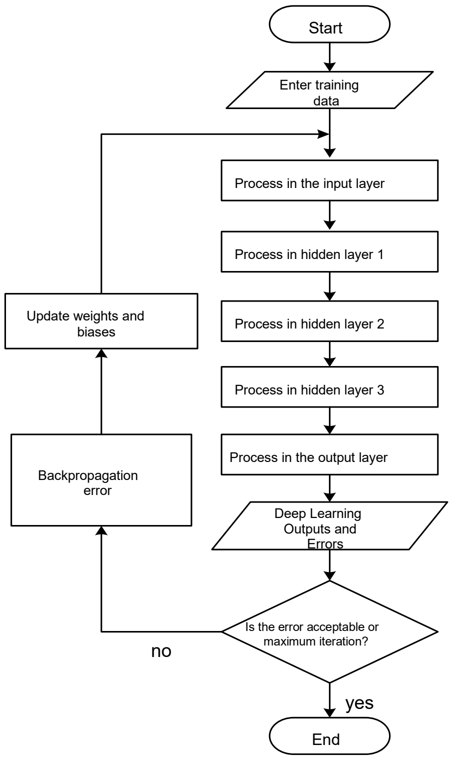

Start
Enter training data
Process in the input layer
Process in hidden layer 1
Process in hidden layer 2
Process in hidden layer 3
Process in the output layer
Deep Learning Outputs and Errors
Is the error acceptable or maximum iteration?
yes
End
no
Backpropagation error
Update weights and biases

Figure 1. Deep Learning training process flow chart

From Figure 1 it can be described as follows. Where the Deep Learning network is trained using training data in the form of pairs of input and output data according to the training data model. In the input layer, the training data is processed with weights and input bias using the ReLu activation function. Proceed to the process in hidden layers 1, 2, and 3, with hidden and biased layer weights using the ReLu activation function. Until the output layer is used the output and bias weights and the sigmoid activation function. Until the Deep Learning output is obtained.

Furthermore, the Deep Learning output is compared with the target or desired output, so a difference or error is obtained. Furthermore, this error is attempted to decrease (decrease) using gradient descent with the Error backpropagation (EBP) method. From the EBP new weights and biases are obtained. With these new weights and biases, the process is repeated for the next iteration. Thus, this process is repeated until the error or the difference between the output and the target is acceptable or the maximum iteration is reached.

Based on the flow chart of the Deep Learning training process in Figure-1, after the error or the difference between the Deep Learning output and the target has been accepted or the maximum iteration has been reached, then the weights and bias of the results of this training are stored. Henceforth, if the Deep Learning network is given new data, it is processed at each layer until the Deep Learning network output is obtained which is called prediction. The results of this prediction will be used later in this study to predict the b-value as a parameter that is believed to be a precursor to an earthquake. Thus, the flow chart of the Deep Learning testing process is as shown in Figure 2.

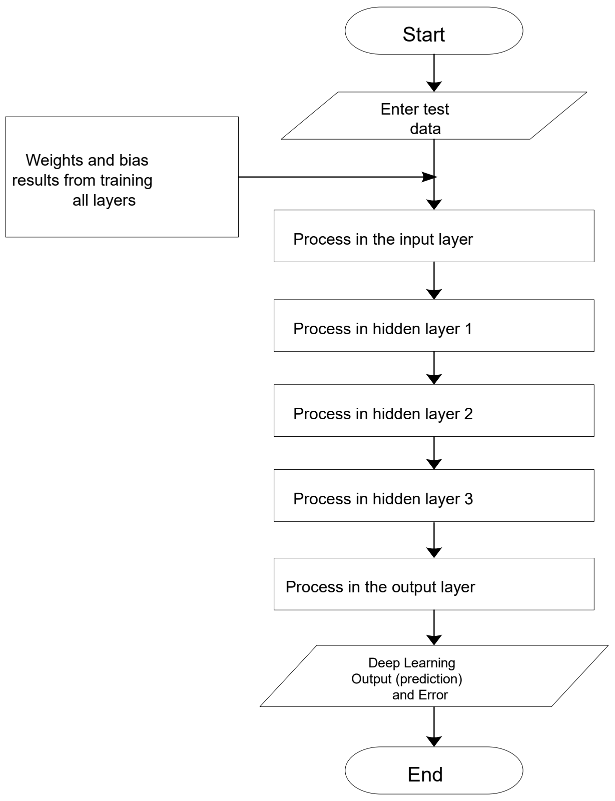

Start
Enter test data
Weights and bias results from training all layers
Process in the input layer
Process in hidden layer 1
Process in hidden layer 2
Process in hidden layer 3
Process in the output layer
Deep Learning Output (prediction) and Error
End

Figure 2. Deep learning testing process flow chart

Furthermore, as previously mentioned, the earthquake prediction program using the Deep Learning method will be created using the Python programming language with the Keras library. The design of the earthquake prediction program with Deep Learning using Keras is designed with the following programming sequence: Data Preparation, Determination of Keras Model, Compile Keras Model, Keras Model Adjustments, Keras Model Evaluation, and Making predictions, as shown in Figure 3.

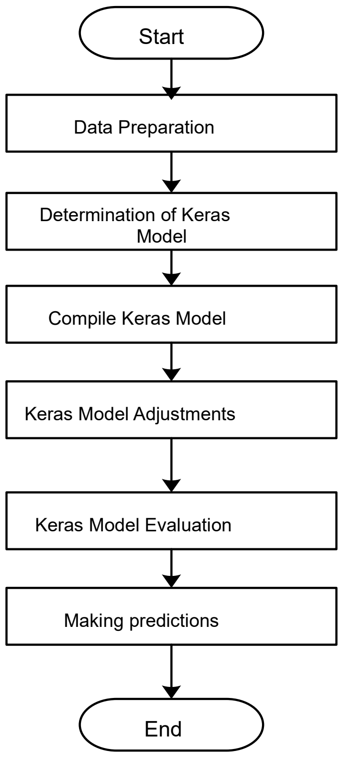


Figure 3. Earthquake prediction programming flowchart based on Deep Learning using Keras

Results and Discussion

For testing python programming for a deep learning-based earthquake prediction system with the Keras library, the results areas in the following description. The deep learning system network training process with 31 inputs and 1 output, using b-value input data from December 1973 to November 2008. The target data or the desired output is data from December 2008 to November 2009. Results the training process using 10,000 iterations, as shown in Figure 4.

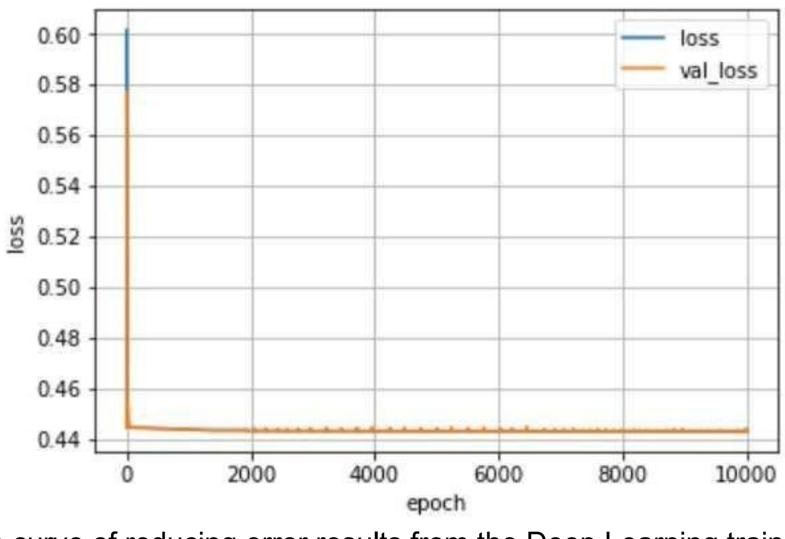


Figure 4. The curve of reducing error results from the Deep Learning training process

From the results of the Deep Learning training process for the b-value prediction system as an earthquake precursor with several iterations of 10,000 times. The results of Mean

Square Error (MSE), Root Mean Square Error (RMSE), Mean Absolute Percentage Error (MAPE), and the percentage of successful predictions are obtained 5.43 x 10-5; 0.00737; 0.80897 and 99.19% respectively. In summary, presented in tabular form, as shown in Table 1.

Table 1. The results of the Deep Learning training process

Number of Iterations	MSE	RMSE	MAPE	Success Percentage (%)
10.000	5,43 x 10-5	0,00737	0,80897	99,19

Furthermore, for testing, tested 5 times. For example in the first test, the Deep Learning network system was given b-value input data from December 1974 to November 2009. It is used to predict the b-value as a precursor to earthquakes for the next year, from December 2009 to November 2010. An example of the prediction results is shown in Figure-5. And as a whole, a series of tests for the prediction of the b-value as an earthquake precursor which was carried out five times the test obtained an average of MSE, RMSE, MAPE, and the percentage of successful predictions were 0.03886; 0.19003; 23.96459, and 77.75%. Completely, each test result and the average test result are presented in Table 2.

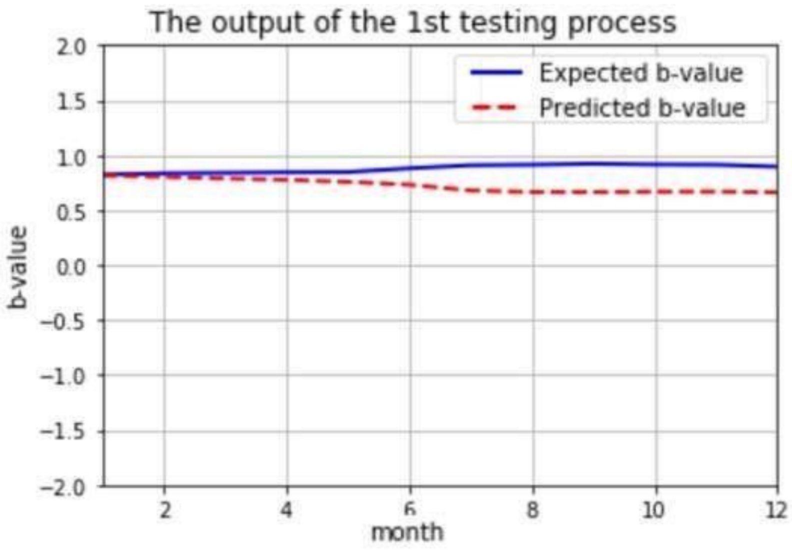
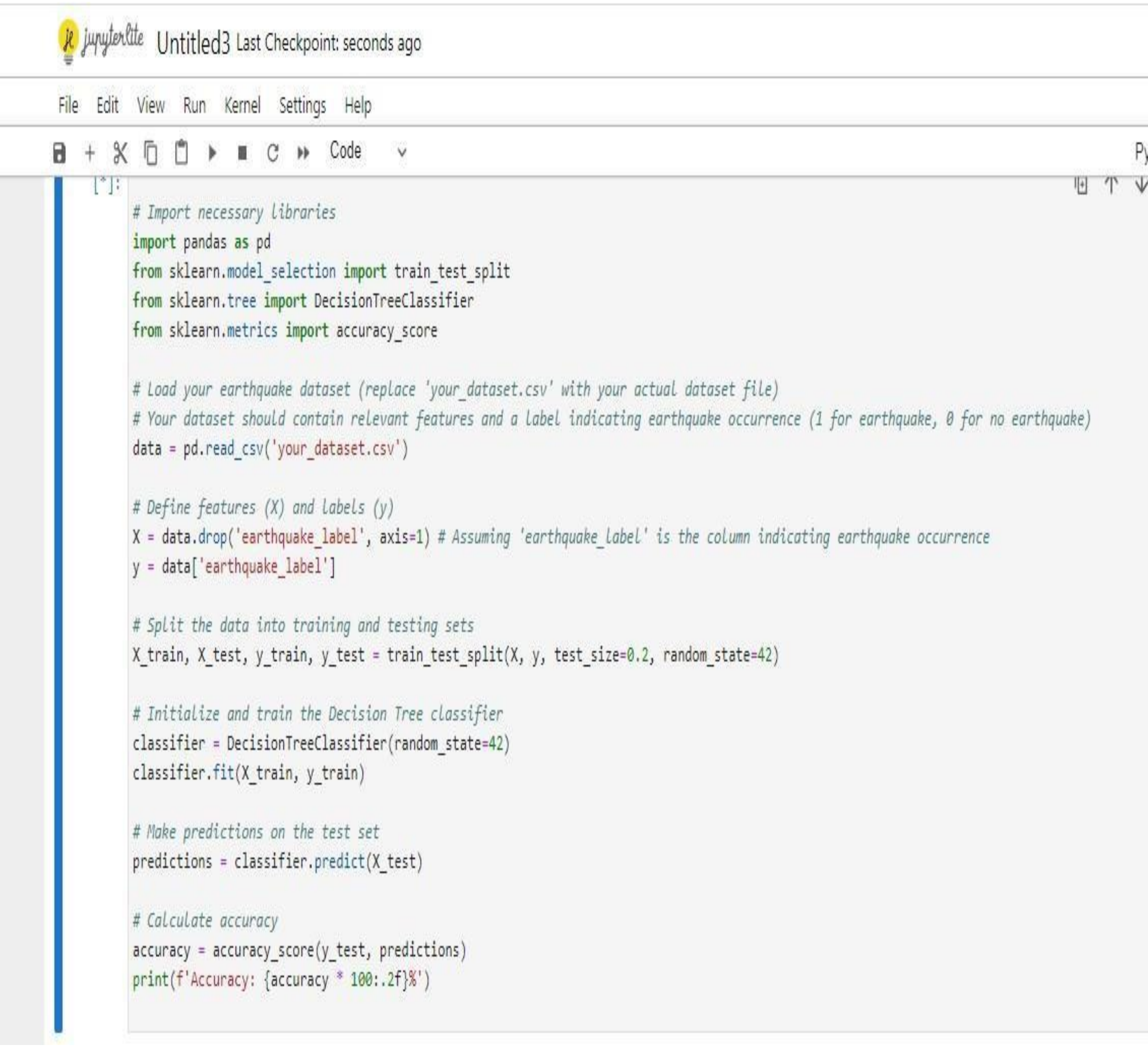


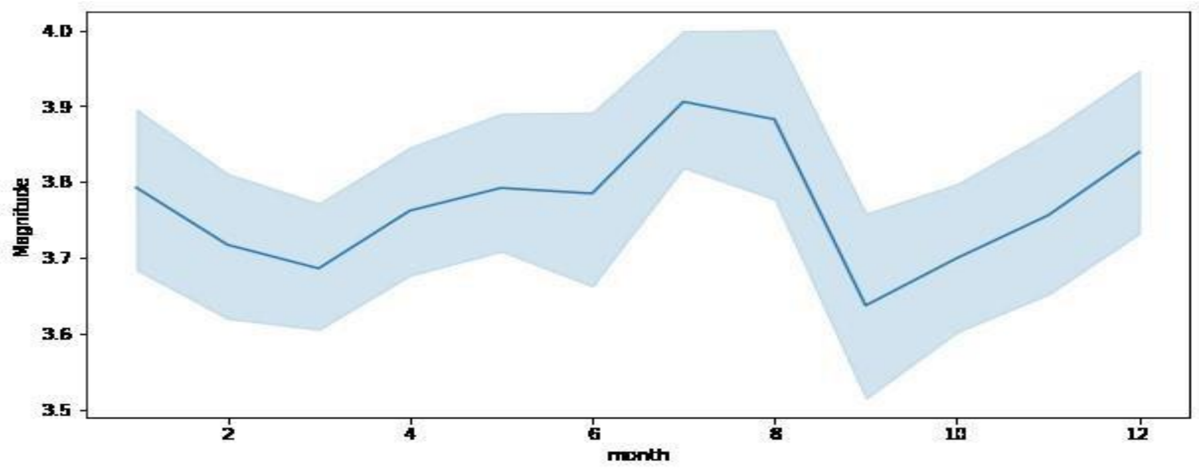
Figure 5. The output of the 1st Deep Learning testing process

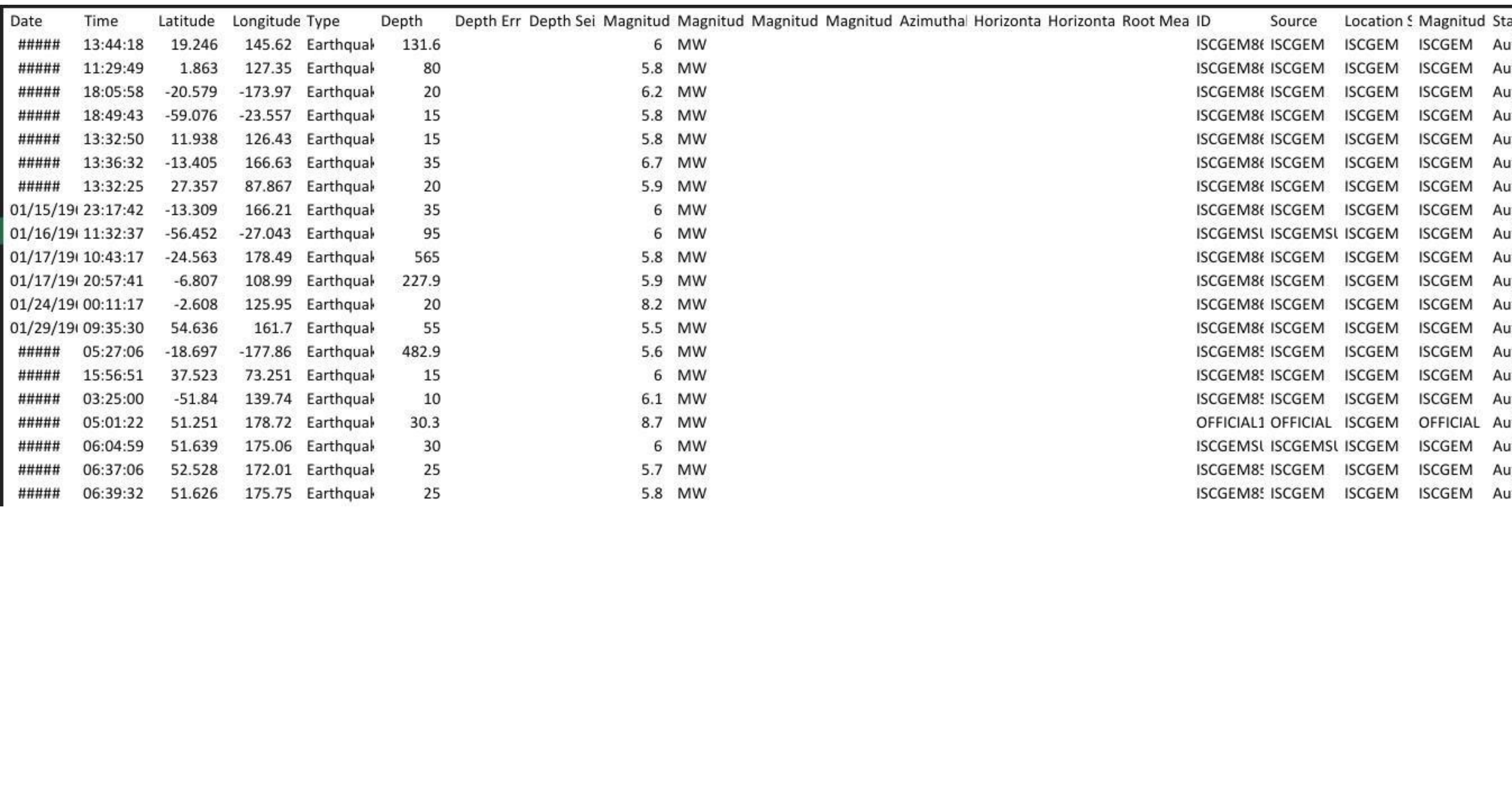
Table 2. The results of the Deep Learning training process

Testing	MSE	RMSE	MAPE	Success Percentage (%)
1	0,03286	0,18128	17,21536	82,78
2	0,05779	0,24039	26,10020	82,46
3	0,01891	0,13752	17,53647	82,46
4	0,06755	0,25991	42,35427	57,65
5	0,01718	0,13106	16,61665	83,38
Average	0,03886	0,19003	23,96459	77,75


jupyterlite
Untitled3 Last Checkpoint: seconds ago
File Edit View Run Kernel Settings Help
Code
[*]:

```python
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load your earthquake dataset (replace 'your_dataset.csv' with your actual dataset file)
# Your dataset should contain relevant features and a label indicating earthquake occurrence (1 for earthquake, 0 for no earthquake)
data = pd.read_csv('your_dataset.csv')

# Define features (X) and labels (y)
X = data.drop('earthquake_label', axis=1) # Assuming 'earthquake_label' is the column indicating earthquake occurrence
y = data['earthquake_label']

# Split the 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)

# Initialize and train the Decision Tree classifier
classifier = DecisionTreeClassifier(random_state=42)
classifier.fit(X_train, y_train)

# Make predictions on the test set
predictions = classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
print(f'Accuracy: {accuracy * 100:.2f}%')
```





Date	Time	Latitude	Longitude	Type	Depth	Depth Err	Depth Sei	Magnitud	Magnitud	Magnitud	Magnitud	Azimuthal	Horizonta	Horizonta	Root Mea	ID	Source	Location S	Magnitud	Sta
#####	13:44:18	19.246	145.62	Earthquak	131.6			6	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Au
#####	11:29:49	1.863	127.35	Earthquak	80			5.8	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Au
#####	18:05:58	-20.579	-173.97	Earthquak	20			6.2	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Au
#####	18:49:43	-59.076	-23.557	Earthquak	15			5.8	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Au
#####	13:32:50	11.938	126.43	Earthquak	15			5.8	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Au
#####	13:36:32	-13.405	166.63	Earthquak	35			6.7	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Au
#####	13:32:25	27.357	87.867	Earthquak	20			5.9	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Au
01/15/19	23:17:42	-13.309	166.21	Earthquak	35			6	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Au
01/16/19	11:32:37	-56.452	-27.043	Earthquak	95			6	MW							ISCGEMSU	ISCGEMSU	ISCGEM	ISCGEM	Au
01/17/19	10:43:17	-24.563	178.49	Earthquak	565			5.8	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Au
01/17/19	20:57:41	-6.807	108.99	Earthquak	227.9			5.9	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Au
01/24/19	00:11:17	-2.608	125.95	Earthquak	20			8.2	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Au
01/29/19	09:35:30	54.636	161.7	Earthquak	55			5.5	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Au
#####	05:27:06	-18.697	-177.86	Earthquak	482.9			5.6	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Au
#####	15:56:51	37.523	73.251	Earthquak	15			6	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Au
#####	03:25:00	-51.84	139.74	Earthquak	10			6.1	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Au
#####	05:01:22	51.251	178.72	Earthquak	30.3			8.7	MW							OFFICIAL1	OFFICIAL	ISCGEM	OFFICIAL	Au
#####	06:04:59	51.639	175.06	Earthquak	30			6	MW							ISCGEMSU	ISCGEMSU	ISCGEM	ISCGEM	Au
#####	06:37:06	52.528	172.01	Earthquak	25			5.7	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Au
#####	06:39:32	51.626	175.75	Earthquak	25			5.8	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Au

Conclusion

Python programming language is reliable enough to solve machine learning and deep learning problems. With the python library used, namely Keras, deep learning programming for the earthquake prediction system is implemented in the following stages: data preparation, determining the Keras model, compiling the Keras model, adjusting the Keras model, evaluating the Keras model, and making a prediction system. From the test results of the earthquake prediction system using the Python programming language, the results are quite satisfactory.