**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**College code: 2108**

**College Name: Kings Engineering College**

**Project Domain: Artificial intelligence**

**Project Title: Earthquake Prediction Model using Python**

**Project mentor: Dr. Bestley Joe**

**Team members:**

**1.Tasleem Fathima (210821106086)**

**2.Ramya. M(210821106056)**

**3.Usha. J(210821106088)**

**4.Ramya. M(210821106055)**

**Introduction:**

An earthquake prediction model using Python is a crucial endeavor in the field of seismology and geophysics. This project aims to harness the power of data and machine learning to forecast seismic events, potentially mitigating their devastating impact. In this model, we will leverage historical seismic data, sensor readings, and various features to create a predictive framework. Through this introduction, we will explore the key components of the model, from data collection and preprocessing to the application of machine learning algorithms for earthquake forecasting. By the end of this project, we aspire to provide a valuable tool that can contribute to early earthquake warning systems and enhance our understanding of seismic activity.

**Tools and Software:**

* Python: Python is the primary programming language for developing the model. It provides a wide range of libraries for data analysis, machine learning, and real-time data processing.
* Jupyter Notebook: Jupyter Notebook is a popular environment for interactive data analysis and code development. It's commonly used for exploring seismic data and developing the model.
* NumPy and SciPy: NumPy and SciPy are essential libraries for scientific and numerical computing in Python. They provide support for array manipulation, linear algebra, and statistical functions.
* Pandas: Pandas is a library for data manipulation and analysis. It's useful for handling and preprocessing seismic data.
* Matplotlib and Seaborn: Matplotlib and Seaborn are libraries for data visualization. They help create informative plots and visualizations for better data interpretation.
* scikit-learn: scikit-learn is a machine learning library that provides tools for building predictive models, including regression and classification algorithms used in earthquake prediction.
* TensorFlow or PyTorch: These deep learning frameworks are essential for developing neural network-based models for earthquake prediction, especially in cases where more complex architectures are required.
* Apache Spark: In cases involving large-scale data processing, Apache Spark can be used for distributed data processing and machine learning tasks.
* SQL and Databases: SQL databases like PostgreSQL or NoSQL databases like MongoDB can be used to store and retrieve seismic data for analysis.
* Geospatial Libraries: Libraries like GeoPandas and Shapely can be used to work with geospatial data, especially for earthquake location and mapping.
* Data Collection Tools: Depending on your data sources, you may need tools for collecting real- time seismic data from sensors and networks.
* Geographical Information Systems (GIS): GIS software can be used to manage, visualize, and analyze geospatial data relevant to earthquake prediction models.
* Sensor Networks: Hardware and software tools for setting up and managing sensor networks to collect real-time seismic data are essential for some prediction models.
* Statistical Software: Statistical packages like R or specialized software for geostatistics may be used for specific data analysis tasks.
* Version Control: Tools like Git and platforms like GitHub are crucial for code version control, collaboration, and reproducibility.
* Cloud Services: Cloud computing platforms like AWS, Google Cloud, or Azure can be used for scalable data processing and machine learning tasks.
* Containerization: Tools like Docker can help package and deploy models and applications consistently across different environments.

**1. DESIGN THINKING AND PRESENT IN FORM OF DOCUMENT**

* Empathize:
* Identify and describe the stakeholders: scientists, emergency responders, local communities, etc.
* Gather their perspectives on current challenges in earthquake prediction.
* Conduct interviews, surveys, and observations to empathize with their needs and concerns.
* Define:
* Summarize the key problems and pain points identified during the Empathize phase.
* Define clear problem statements related to earthquake prediction.
* Ideate:
* Brainstorm potential solutions to address the defined problems.
* Encourage creativity and generate a wide range of ideas.
* Use techniques like mind mapping, brainstorming sessions, and design workshops.
* Prototype:
* Select the most promising ideas from the Ideate phase.
* Create prototypes or concept models of the earthquake prediction system.
* Focus on creating a tangible representation of the solution.
* Test:
* Test the prototypes with stakeholders, seeking their feedback.
* Refine and iterate the prototypes based on feedback.
* Ensure the solution aligns with the identified problems and needs.
* Implement:
* Develop a detailed plan for building the earthquake prediction model.
* Allocate necessary resources and set timelines.
* Collaborate with experts in geophysics, data science, and technology.
* Monitor and Evaluate:
* Once the model is implemented, continuously monitor and evaluate its performance. - Collect data on its accuracy and reliability.
* Make improvements as necessary based on real-world results.
* **DESIGN THINKING INTO INNOVATION**
* Download dataset from Kaggle using the below link

Link: https://[www.kaggle.com/datasets/usgs/earthquake-database](http://www.kaggle.com/datasets/usgs/earthquake-database)

* Go through the dataset and filter as per needed
* Extract the libraries that are needed work with the dataset for earthquake prediction.
* Using **Pandas** library is the most helpful feature in python to handle with datasets.
* Using **Sklearn** library in python is a best library for prediction type machine learning models which are pre build in it.
* Split the data into train and test data so that the model uses certain data for training purposes and after training the model can be evaluated using the test data. This can be done through the **train\_test\_split()** function from **sklearn**
* After the model is trained, find it’s accuracy using functions like **mean\_squared\_error**, accuracy or any other similar approaches
* **BUILD LOADING AND PREPROCESSING THE DATASET**
* **Install Required Libraries:**

Ensure you have the necessary libraries installed by running these commands in your terminal or Jupyter Notebook:

```bash

pip install numpy pandas scikit-learn matplotlib kaggle

```

* **Kaggle API Setup:**

You need to set up your Kaggle API credentials. Follow the Kaggle API setup guide to generate an API key file and place it in your user directory.

* **Download the Dataset:**

Download the dataset using the Kaggle API. You can use the following code:

```python import kaggle

# Define your Kaggle API credentials kaggle.api.authenticate(api\_key="your\_api\_key\_here")

# Download the dataset kaggle.api.dataset\_download\_files(

dataset="andrewmvd/earthquakes", path="./",

unzip=True

)

```

* **Load and Preprocess the Data:**

df = pd.read\_csv('../input/earthquakes-for-ml-prediction-ne w-version/silver.csv')

df[:5]

**Output:**

* **Building an Earthquake Prediction Model:**

**Covert Date:**



df['date'] = pd.to\_datetime(df['date'])

Magnitude analysis:

df['mag\_rounded\_down'] = df['mag'].astype(int) df.groupby('mag\_rounded\_down').agg({'id': 'count'})

**Output:**

|  |  |
| --- | --- |
| 3 | 211629 |
| 4 | 372804 |
| 5 | 73634 |
| 6 | 6135 |
| 7 | 634 |
| 8 | 37 |

**Calculate energy :**

df['energy'] = 5.24 df['energy'] += 1.44

\* df['mag'] df['energy'] = np.power(10, df['energy'])

Count through years:

plt.hist(df['date'], bins = 500);



* **Evaluate and Fine-Tune:**

Evaluate the model's performance and fine-tune it as needed. You may want to try different algorithms and hyperparameters to improve prediction accuracy.

* **Deploy and Use the Model:**

Once your model is trained and evaluated, you can deploy it for earthquake prediction using new data.

Remember that predicting earthquakes is a complex and challenging task, and the above steps provide a basic framework. More sophisticated models and domain expertise may be necessary for accurate predictions.

EXPLORATORY ANALYSIS

To begin this exploratory analysis, first import libraries and define functions for plotting the data using `matplotlib`. Depending on the data, not all plots will be made. (Hey, I'm just a simple kerneling bot, not a Kaggle Competitions Grandmaster!)

**['database.csv']**

Read the data from csv and also columns which are necessary for the model and the column which needs to be predicted.

data = pd.read\_csv("../input/database.csv") data.head()

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

final\_data = data.drop(['Date', 'Time'], axis=1)

final\_data = final\_data[final\_data.Timestamp != 'ValueError'] final\_data.head()

**Visualization**

Here, all the earthquakes from the database in visualized on to the world map which shows clear representation of the locations where frequency of the earthquake will be more.

**Splitting the Data:**



Firstly, split the data into Xs and ys which are input to the model and output of the model respectively. Here, inputs are TImestamp, Latitude and Longitude and outputs are Magnitude and Depth. Split the Xs and ys into train and test with validation. Training dataset contains 80% and Test dataset 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)

reg.score(X\_test, y\_test)

best\_fit.score(X\_test, y\_test)

**Neural Network model**

In the above case it was more kind of linear regressor where the predicted values are not as expected. So, Now, we build the neural network to fit the data for training set. Neural Network consists of three Dense layer with each 16, 16, 2 nodes and relu, relu and softmax as activation function.

Using TensorFlow backend.

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)



In this, we define the hyperparameters with two or more options to find the best fit.

Here, we find the best fit of the above model and get the mean test score and standard deviation of the best fit model.

The best fit parameters are used for same model to compute the score with training data and testing data.

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 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 2/20

18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 3/20

18727/18727 [==============================] - 4s 228us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 4/20

18727/18727 [==============================] - 4s 222us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 5/20

18727/18727 [==============================] - 5s 262us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 6/20

18727/18727 [==============================] - 4s 223us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 7/20

18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 8/20

18727/18727 [==============================] - 4s 224us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 9/20

18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 10/20

18727/18727 [==============================] - 4s 224us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 11/20

18727/18727 [==============================] - 4s 221us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 12/20

18727/18727 [==============================] - 4s 231us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 13/20

18727/18727 [==============================] - 5s 248us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 14/20

18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 15/20

18727/18727 [==============================] - 4s 223us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 16/20

18727/18727 [==============================] - 4s 222us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 17/20

18727/18727 [==============================] - 4s 225us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 18/20

18727/18727 [==============================] - 4s 219us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 19/20

18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

Epoch 20/20

18727/18727 [==============================] - 5s 258us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc

: 0.9242

<keras.callbacks.History at 0x78dfa2107ef0>

[test\_loss, test\_acc] = model.evaluate(X\_test, y\_test)

print("Evaluation result on Test Data : Loss = **{}**, accuracy = **{}**".format(test\_loss, test\_acc))

4682/4682 [==============================] - 0s 29us/step

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

We see that the above model performs better but it also has lot of noise (loss) which can be neglected for prediction and use it for furthur prediction.

The above model is saved for furthur prediction

model.save('earthquake.h5')



ADVANTAGES:

* Data-Driven Insights: Python's powerful data analysis and visualization libraries, such as Pandas, NumPy, and Matplotlib, allow for in-depth analysis of historical seismic data, providing valuable insights into earthquake patterns and trends.
* Machine Learning: Python's extensive ecosystem for machine learning, including libraries like scikit-learn and TensorFlow, enables the development of predictive models that can learn from past seismic events and make forecasts based on patterns and features in the data.
* Real-time Monitoring: Python's flexibility makes it suitable for real-time data processing and analysis, which is crucial for monitoring seismic activity and issuing timely alerts.
* Scalability: Python's ability to work with big data technologies, such as Apache Spark and Hadoop, allows for scaling the model to handle large volumes of seismic data.
* Open Source Community: Python benefits from a vast and active open-source community. This means access to numerous libraries, resources, and contributions for earthquake prediction research.
* Visualization Tools: Python's data visualization libraries, like Seaborn and Plotly, assist in creating informative plots and visualizations for easier interpretation of seismic data and model results.
* Cross-Platform Compatibility: Python is cross-platform, meaning that models developed can be deployed on various systems and platforms, making it adaptable for different monitoring and prediction needs.
* Integration with Sensor Networks: Python can be integrated with sensor networks and IoT devices to collect and process real-time seismic data, enhancing the model's accuracy and efficiency.
* Research and Collaboration: Python's widespread use in the scientific and research communities facilitates collaboration and sharing of earthquake prediction models, fostering advancements in the field.
* Early Warning Systems: The model can be integrated into early warning systems, potentially providing valuable lead time to prepare for seismic events and mitigate their impact.

**DISADVANTAGES**:

* \*Data Quality and Quantity\*: The accuracy of the model heavily depends on the quality and quantity of available seismic data. Incomplete or inaccurate data can lead to unreliable predictions.
* Complexity: Earthquake prediction is a complex and multifaceted problem. Developing an accurate model involves intricate geophysical and seismological knowledge, which may require collaboration with domain experts.
* Prediction Uncertainty: Earthquake prediction inherently carries uncertainty. Pythonbased models can provide forecasts, but they may not offer precise predictions and could lead to false alarms.
* Resource Intensive: Handling and processing large seismic datasets, especially in realtime applications, can be resource-intensive. High-performance computing and substantial storage capacity may be required.
* Model Tuning: Fine-tuning machine learning models in Python can be a time- consuming and iterative process. Proper hyperparameter selection and model evaluation are crucial for meaningful results.

**CONCLUSION**:

In conclusion, the development of an earthquake prediction model using Python represents a significant and promising endeavor. This project leverages the power of data analysis, machine learning, and real-time capabilities to contribute to our understanding of seismic activity and enhance disaster preparedness. While acknowledging the inherent challenges and uncertainties in earthquake prediction, this model offers the potential for early warning and the mitigation of the devastating impacts of seismic events. By harnessing Python's versatile ecosystem, we can analyze historical seismic data, detect patterns, and develop predictive frameworks, all while fostering collaboration, research, and innovation in the fields of seismology and data science. The pursuit of earthquake prediction models remains a critical step towards a safer and more resilient world in the face of this natural disaster.