EARTHQUAKE PREDICTION MODEL USING PYTHON

PHASE 4 : Development part 2

- ❖ In this section continue building the project performing different activities like feature engineering, model training, evaluation etc as per the instructions in the project.

Random Forest Algorithm :

- ❖ It is a type of machine learning algorithm that is very famous nowadays. It generates a random decision tree and combines it into a single forest.
- ❖ The tree's "root" node represents the entire predictor space. The final division of the predictor space is made up of the "terminal nodes," which are nodes that are not split.
- ❖ Depending on the value of one of the predictor variables, each nonterminal node divides into two descendant nodes, one on the left and one on the right.
- ❖ If a continuous predictor variable is smaller than a split point, the points to the left will be the smaller predictor points, and the points to the right will be the larger predictor points.
- ❖ It is also a classification technique that uses ensemble learning. The random forest generates a root node feature by randomly dividing, which is the primary distinction between it and the decision tree.

Support Vector Classifier

- ❖ There is a computer algorithm known as a support vector machine (SVM) that learns to name objects.

- For instance, by looking at hundreds or thousands of reports of both fraudulent and legitimate credit card activity, an SVM can learn to identify fraudulent credit card activity.
- A vast collection of scanned photos of handwritten zeros, ones, and other numbers can also be used to train an SVM to recognize handwritten numerals.

Project Prerequisites

The requirement for this project is Python 3.6 installed on your computer. I have used Jupyter notebook for this project. You can use whatever you want.

The required modules for this project are –

- Numpy(1.22.4) – pip install numpy
- Sklearn(1.1.1) – pip install sklearn
- Pandas(1.5.0) – pip install pandas

That's all we need for our earthquake prediction project.

Steps to Implement

1. Import the modules and all the libraries we would require in this project.

```
import numpy as np#importing the numpy module
import pandas as pd#importing the pandas module
from sklearn.model_selection import train_test_split#importing the train test split
module
import pickle #import pickle
from sklearn import metrics #import metrics
from sklearn.ensemble import RandomForestClassifier#import the Random Forest
Classifier
```

2. Here we are reading the dataset and we are creating a function to do some data processing on our dataset. Here we are using the numpy to convert the data into an array.

```
dataframe= pd.read_csv("dataset.csv")#here we are reading the dataset
dataframe= np.array(dataframe)#converting the dataset into an numpy array
print(dataframe)#printing the dataframe
```

3. Here we are dividing our dataset into X and Y where x is the independent variable whereas y is the dependent variable. Then we are using the test train split function to divide the X and Y into training and testing datasets. We are taking the percentage of 80 and 20% for training and testing respectively.

```
x_set = dataframe[:, 0:-1]#getting the x dataset
y_set = dataframe[:, -1]#getting the y dataset
y_set = y_set.astype('int')#converting the y_set into int
x_set = x_set.astype('int')#converting the x_set into int
x_train, x_test, y_train, y_test = train_test_split(x_set, y_set, test_size=0.2,
random_state=0)
```

4. Here we are creating our RandomForestClassifier and we are passing our training dataset to our model to train it. Also then we are passing our testing dataset to predict the dataset.

```
Random_forest_classifier = RandomForestClassifier()#creating the model
random_forest_classifier.fit(x_train, y_train)#fitting the model with training dataset
y_pred = random_forest_classifier.predict(X_test)#predicting the result using test set
print(metrics.accuracy_score(y_test, y_pred))#printing the accuracy score
```

5. In this piece of code, we are creating our instance for Gradient Boosting Classifier. The maximum Depth of this Gradient Boosting algorithm is 3.

After creating the instance, we pass our training data to the classifier to fit our training data into the algorithm. This is a part of training.

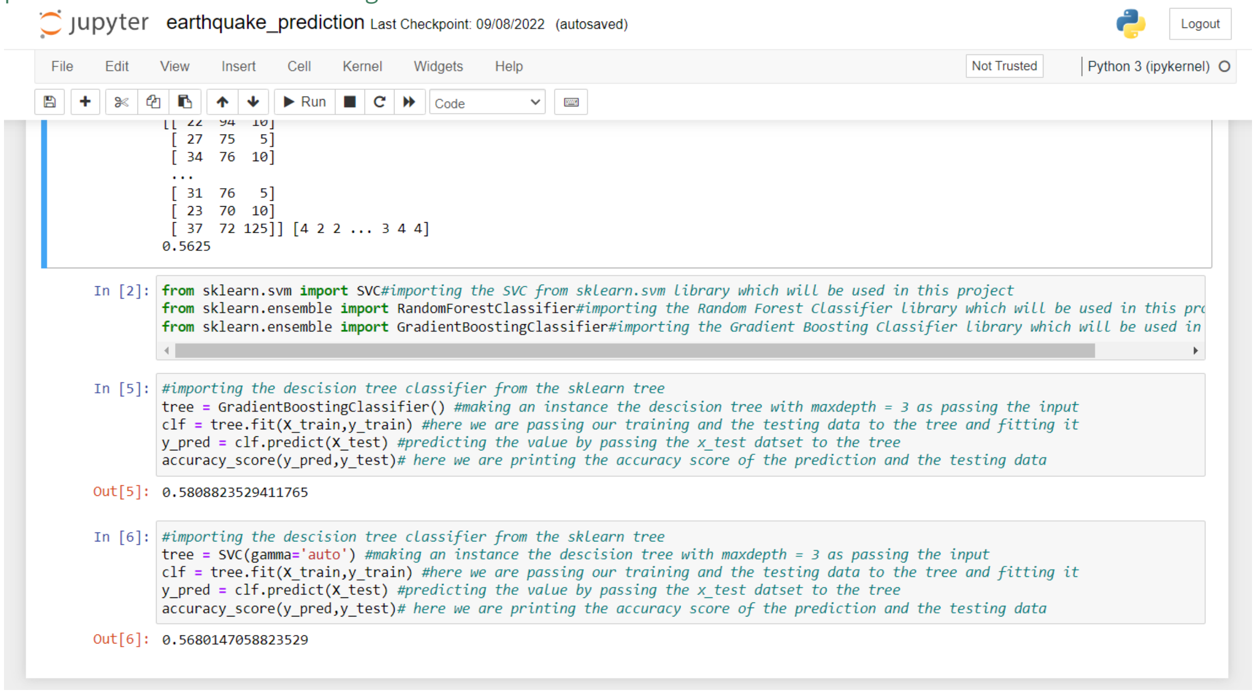
```
#importing the descision tree classifier from the sklearn tree
tree = GradientBoostingClassifier() #making an instance the descision tree with
maxdepth = 3 as passing the input
clf = tree.fit(X_train,y_train) #here we are passing our training and the testing data
to the tree and fitting it
y_pred = clf.predict(X_test) #predicting the value by passing the x_test datset to the
tree
accuracy_score(y_pred,y_test)# here we are printing the accuracy score of the
prediction and the testing data
```

6. Here, we are doing the same thing as above.

After training the model, we pass the testing data to our model and predict the accuracy score using the accuracy score function.

```
#importing the descision tree classifier from the sklearn tree
tree = SVC(gamma='auto') #making an instance the support vector tree
clf = tree.fit(X_train,y_train) #here we are passing our training and the testing data
to the tree and fitting it
```

```
y_pred = clf.predict(X_test) #predicting the value by passing the x_test datset to the
tree
accuracy_score(y_pred,y_test)# here we are printing the accuracy score of the
prediction and the testing data
```


jupyter earthquake_prediction Last Checkpoint: 09/08/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

```
[[ 22  94  10]
 [ 27  75   5]
 [ 34  76  10]
 ...
 [ 31  76   5]
 [ 23  70  10]
 [ 37  72 125]] [4 2 2 ... 3 4 4]
0.5625
```

```
In [2]: from sklearn.svm import SVC#importing the SVC from sklearn.svm library which will be used in this project
        from sklearn.ensemble import RandomForestClassifier#importing the Random Forest Classifier library which will be used in this pro
        from sklearn.ensemble import GradientBoostingClassifier#importing the Gradient Boosting Classifier library which will be used in

In [5]: #importing the descision tree classifier from the sklearn tree
        tree = GradientBoostingClassifier() #making an instance the descision tree with maxdepth = 3 as passing the input
        clf = tree.fit(X_train,y_train) #here we are passing our training and the testing data to the tree and fitting it
        y_pred = clf.predict(X_test) #predicting the value by passing the x_test datset to the tree
        accuracy_score(y_pred,y_test)# here we are printing the accuracy score of the prediction and the testing data

Out[5]: 0.5808823529411765

In [6]: #importing the descision tree classifier from the sklearn tree
        tree = SVC(gamma='auto') #making an instance the descision tree with maxdepth = 3 as passing the input
        clf = tree.fit(X_train,y_train) #here we are passing our training and the testing data to the tree and fitting it
        y_pred = clf.predict(X_test) #predicting the value by passing the x_test datset to the tree
        accuracy_score(y_pred,y_test)# here we are printing the accuracy score of the prediction and the testing data

Out[6]: 0.5680147058823529
```

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. `In [16]:`

```
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
```

```
Using TensorFlow backend.
```

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

In [17]:

```
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)
```

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

In [18]:

```
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': 'squared_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': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.000000 (0.000000) with: {'activation': 'relu', 'batch_size': 10,
'epochs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer':
'Adadelta'}
```

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

In [19]:

```
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'])
```

In [20]:

```
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
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.

In [22]:

```
model.save('earthquake.h5')
```