ANN PROJECT REPORT

Submitted By Kamini Sharma

23/PHDMCG/501

Table of Contents

1. Introduction

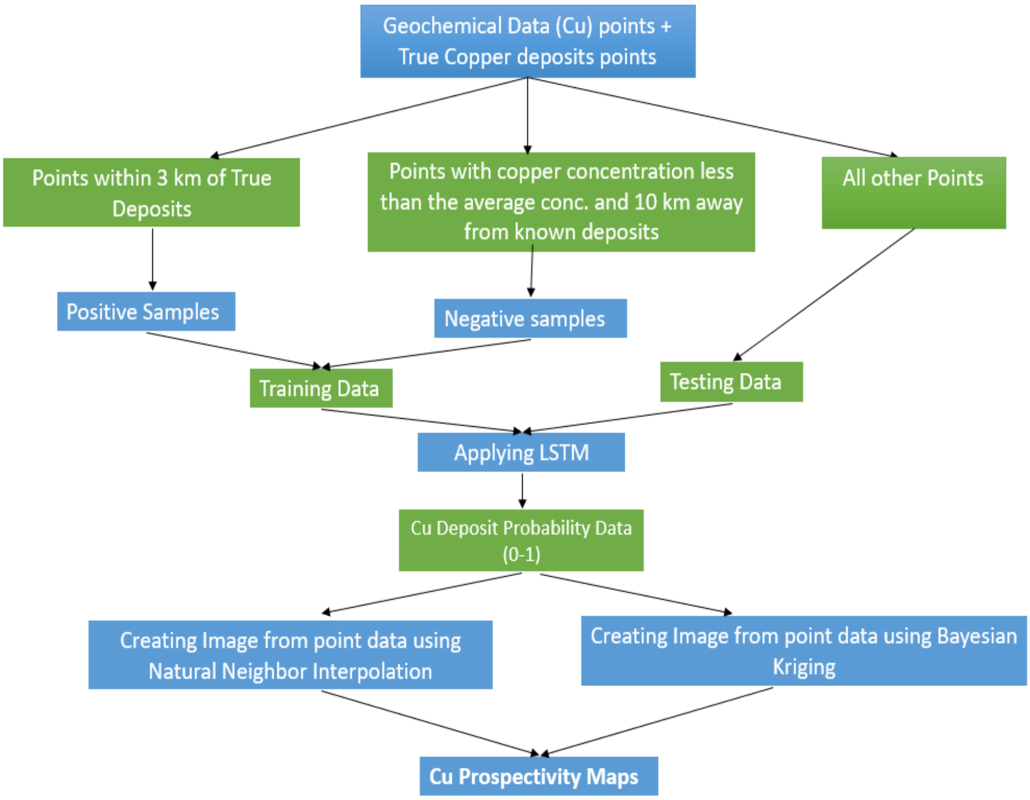
Problem Statement- Creating Copper Deposit prospectivity Map for Churu area, Rajasthan.

The area already hosts Copper mines, a recent news articles stated this district to be prospective of more exploration by Geological Survey of India. As district has area of 13,858 km², a prospectivity map will help in pinpointing the exploration to limited areas of high probability of copper. Although such maps require various inputs, we will explore creating this map by one but important parameter- Geochemical concentration of copper in its sediments on the surface using ANN and GIS software.

2. Dataset

Data includes Geochemical Point data for the district. It was collected by GSI and was acquired from Bhukosh portal. Data has location and copper concentration in surface sediments, 6515 points covering most of the part of the district churu. Along with that we have locations of current copper deposits/mines.

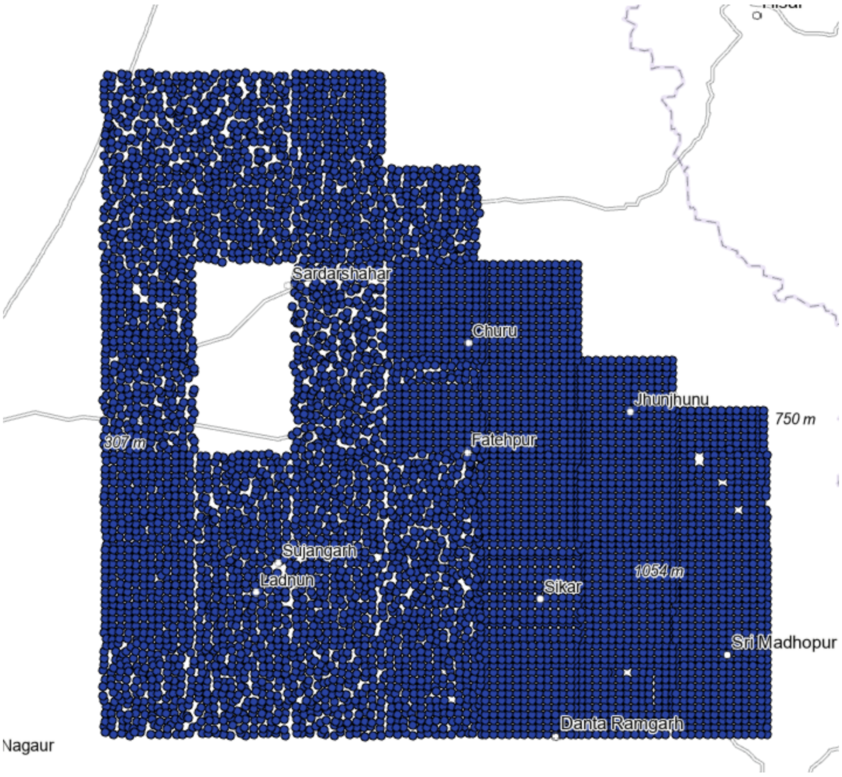
3. Methodology



Steps followed;

I. Collected Copper Conc. Data from GSI.

This data was acquired from Geological Survey of India portal-Bhukosh. It shows the geochemical concentration of copper in the sediments in and near Churu District. Data is displayed as points in ArcGIS pro Software.



II. Created Training and Testing data.

Using Points **within 3 Km** of a true deposit of copper- **Positive samples** were created. Using Points with **copper concentration less than the average concentration** of the area and **more 10 km away** from true copper deposit- **Negative samples** were created.

Both Cu conc. samples with value (gridcode) 1 for positive and 0 for negative were combined to form training data with 4566 values.

FID	LONGITUDE	LATITUDE	CU	gridcode
1	74.612944	27.257528	8	0
2	74.713111	27.257889	10	0
3	74.817419	27.257966	10	0
4	74.651556	27.258167	9	0
5	74.588111	27.258444	8	0
6	74.918433	27.258629	4	0
7	74.214315	27.258745	2	0
8	74.172756	27.258766	2	0
9	75.2601	27.259	10	0
10	75.2803	27.259	7	0
11	75.3005	27.259	6	0
12	75.3207	27.259	8	0

Training Data

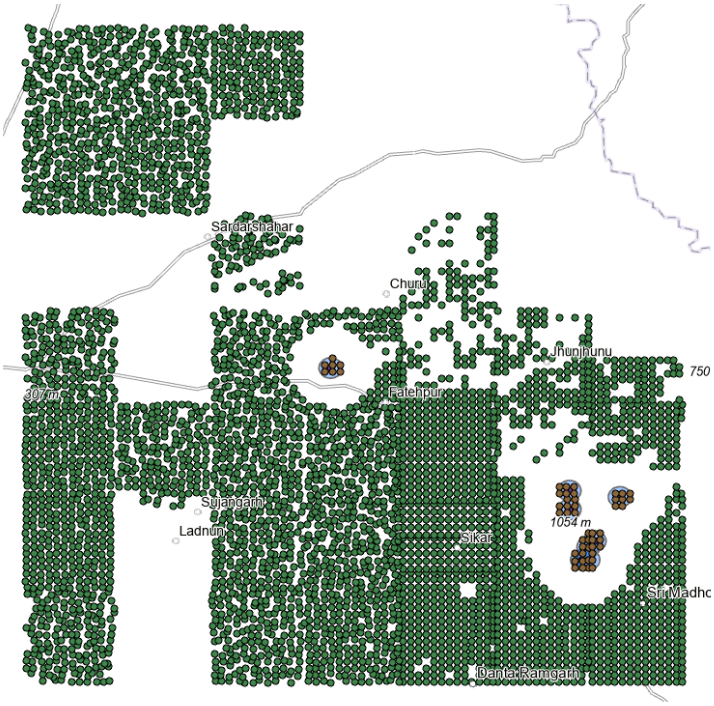
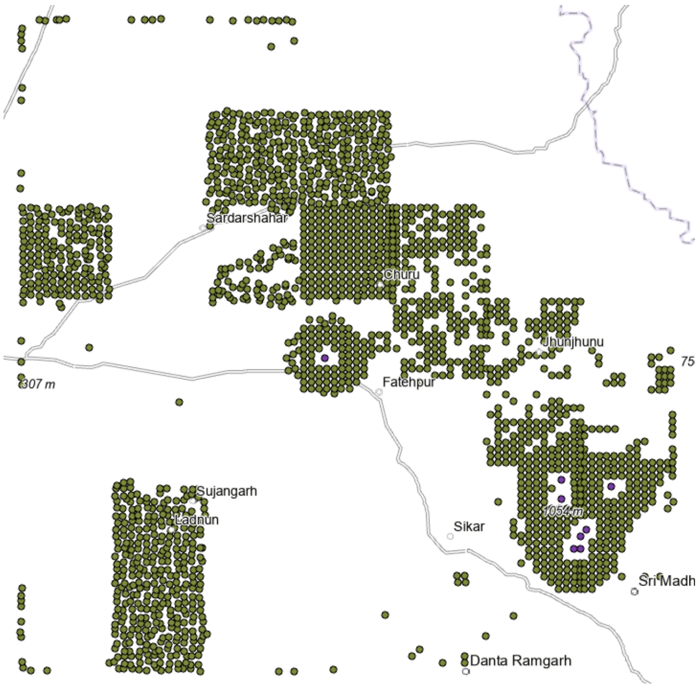


Image Showing Training data used. Green points are negative points and Brown points are positive copper samples present 3 km of copper deposit represented by buffer zone.

Rest of the available data for the area was used for testing i.e. 1986 points shown in image below.

OID	LONGITUDE	LATITUDE	CU
0	74.6935	27.254	10
1	74.484864	27.255431	41
2	74.734639	27.255861	6
3	74.378561	27.256117	50
4	74.340256	27.256969	43
5	74.0757	27.25779	3
6	74.031874	27.258502	1
7	74.400303	27.258911	40
8	74.320633	27.260297	48
9	74.839854	27.260882	17
10	74.266008	27.261272	46
11	74.279311	27.262275	47
12	74.438481	27.262581	42
13	74.301478	27.262708	47

Testing Data



III. Code Created in Google collab and used to get Cu Deposit (Prospectivity) prediction data.

1. Importing the libraries.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
from sklearn.metrics import accuracy_score
```

2. Importing the training and testing set.

```
dataset_test = pd.read_csv('/content/cu_test.csv')
test = dataset_test.sample(frac=1)
print(test)
dataset_train = pd.read_csv('/content/cu_Train.csv')
df = dataset_train.sample(frac=1)
print(df)
```

```
       OID  LONGITUDE   LATITUDE      CU
220    220  75.570856  27.509025   5.000
265    265  75.423769  27.544424   6.000
438    438  75.550683  27.671453  77.000
1409  1409  75.153000  28.403000  17.300
1429  1429  74.214515  28.404369  40.000
...    ...        ...        ...     ...
1701  1701  74.567906  28.561325  49.000
1150  1150  74.923290  28.277070  73.032
1684  1684  74.590406  28.546050  52.000
856    856  75.132270  28.081210  18.871
1477  1477  75.031000  28.439000  29.000

[1986 rows x 4 columns]
       FID  LONGITUDE   LATITUDE    CU  gridcode
2336  2337  74.975028  27.872250  11.0         0
1602  1603  74.549758  27.671033   5.0         0
3836  3837  74.057972  28.614611   1.0         0
3460  3461  74.046250  28.225031  12.0         0
3038  3039  74.963560  28.045120   0.5         0
...    ...        ...        ...   ...       ...
724    725  74.096660  27.424080   3.0         0
4485  4486  74.196714  28.988200   6.0         0
1000  1001  75.361270  27.493650   8.0         0
2200  2201  74.341200  27.844600  11.0         0
1138  1139  75.130000  27.530000   9.0         0

[4566 rows x 5 columns]
```

3. Reshaping and filtering data.

```
test_set = test.iloc[:, 3].values
test.info()
test_set.shape

training_set = df.iloc[:, 3:].values
df.info()
training_set.shape
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1986 entries, 220 to 1477
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   OID        1986 non-null   int64  
 1   LONGITUDE  1986 non-null   float64
 2   LATITUDE   1986 non-null   float64
 3   CU         1986 non-null   float64
dtypes: float64(3), int64(1)
memory usage: 77.6 KB
<class 'pandas.core.frame.DataFrame'>
Index: 4566 entries, 2336 to 1138
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   FID        4566 non-null   int64  
 1   LONGITUDE  4566 non-null   float64
 2   LATITUDE   4566 non-null   float64
 3   CU         4566 non-null   float64
 4   gridcode   4566 non-null   int64  
dtypes: float64(3), int64(2)
memory usage: 214.0 KB
(4566, 2)
```

4. Feature Scaling

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)
training_set_scaled.shape
test_set_reshaped = np.array(test_set).reshape(-1, 1)
fc = MinMaxScaler(feature_range = (0, 1))
test_set_scaled = fc.fit_transform(test_set_reshaped)
test_set_scaled.shape
```

```
(1986, 1)
```

5. Creating a data structure with 60 timesteps and 1 output. This is to retain the memory of previous values in predicting new values to capturing the continuous pattern in a spatial data.

```
X_train = []
y_train = []
for i in range(60, 4566):
    X_train.append(training_set_scaled[i-60:i, 0])
    y_train.append(training_set_scaled[i, 1])
X_train, y_train = np.array(X_train), np.array(y_train)
X_train.shape

#y_train.shape
#len(y_train)
#len(X_train)
X_test = []
for i in range(60, 1986):
    X_test.append(test_set_scaled[i-60:i, 0])
Y_test = np.array(X_test)
Y_test.shape
```

```
(1926, 60)
```

6. Reshaping

```
trainifyx = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
trainifyx.shape
#X_train.shape[1]
#X_train.shape[0]
testingy = np.reshape(Y_test, (Y_test.shape[0], Y_test.shape[1], 1))
testingy.shape
```

```
(1926, 60, 1)
```

7. Building and Training the RNN

```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-pac
```

Importing the Keras libraries and packages

```
import tensorflow as tf
from tensorflow import keras
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping
```

```
early_stopping = EarlyStopping(monitor='loss', patience=4)
```

8. Initialising the RNN

```
regressor = Sequential()
# regressor is an object of squential class and
#represents a sequeance of layers
```

9. Adding the first LSTM layer and some Dropout regularisation

```
regressor.add(LSTM(units = 32, return_sequences = True, input_shape = (X_train.shape[1],
regressor.add(Dropout(0.3))
```

10. Adding the output layer

```
regressor.add(Dense(units = 1))
```

11. Compiling the RNN

```
[86] learning_rate = 0.0001
```

```
[87]
     optimizer = tf.keras.optimizers.Adam(learning_rate = learning_rate)
     regressor.compile(optimizer = optimizer, loss = 'mean_squared_error')
```

12. Fitting the RNN to the Training set

```
regressor.fit(trainifyx, y_train, epochs = 20 , batch_size = 32, callbacks= [early_stopping])
```

```
Epoch 1/20
141/141 [==============================] - 7s 35ms/step - loss: 0.0305
Epoch 2/20
141/141 [==============================] - 4s 26ms/step - loss: 0.0273
Epoch 3/20
141/141 [==============================] - 4s 26ms/step - loss: 0.0248
Epoch 4/20
141/141 [==============================] - 5s 33ms/step - loss: 0.0227
Epoch 5/20
141/141 [==============================] - 4s 30ms/step - loss: 0.0210
Epoch 6/20
141/141 [==============================] - 4s 27ms/step - loss: 0.0195
Epoch 7/20
141/141 [==============================] - 4s 30ms/step - loss: 0.0184
Epoch 8/20
141/141 [==============================] - 5s 33ms/step - loss: 0.0174
Epoch 9/20
141/141 [==============================] - 4s 28ms/step - loss: 0.0166
Epoch 10/20
141/141 [==============================] - 4s 31ms/step - loss: 0.0160
Epoch 11/20
141/141 [==============================] - 4s 31ms/step - loss: 0.0155
Epoch 12/20
141/141 [==============================] - 4s 26ms/step - loss: 0.0151
Epoch 13/20
141/141 [==============================] - 4s 26ms/step - loss: 0.0148
Epoch 14/20
141/141 [==============================] - 5s 34ms/step - loss: 0.0145
Epoch 15/20
141/141 [==============================] - 4s 26ms/step - loss: 0.0143
Epoch 16/20
141/141 [==============================] - 4s 26ms/step - loss: 0.0142
Epoch 17/20
141/141 [==============================] - 5s 33ms/step - loss: 0.0141
Epoch 18/20
141/141 [==============================] - 4s 26ms/step - loss: 0.0140
Epoch 19/20
141/141 [==============================] - 4s 26ms/step - loss: 0.0139
Epoch 20/20
141/141 [==============================] - 5s 32ms/step - loss: 0.0139
<keras.src.callbacks.History at 0x7b196276e2f0>
```

13. Predicting Cu deposit from training, testing data.

```
train_predict = regressor.predict(trainifyx)
```

```
141/141 [==============================] - 2s 8ms/step
```

```
test_predict = regressor.predict(testingy)
```

```
61/61 [==============================] - 1s 9ms/step
```

```
train_predict
```

```
array([[[0.01517234],
        [0.0156559 ],
        [0.01581563],
        ...,
        [0.01580618],
        [0.01581624],
        [0.01583586]],

       [[0.01515354],
        [0.01563596],
        [0.01584225],
        ...,
        [0.01581624],
        [0.01583586],
        [0.01583418]],

       [[0.015141  ],
        [0.01566551],
        [0.01581782],
        ...,
        [0.01583586],
        [0.01583418],
        [0.01582726]],

       ...,

       [[0.01518086],
        [0.01564985],
        [0.01583206],
        ...,
        [0.01580532],
```

14. Reshaping and saving predictions in csv.

```
train_pre = np.reshape(train_predict, (4506, 60))
test_pre = np.reshape(test_predict, (1926, 60))
rup = train_pre[:, 0]
yes = test_pre[:, 0]

you = train_pre[4505, :]
we= you.reshape(60,1)
output = np.append(rup, we)

yes2 = test_pre[1925, :]
vee = yes2.reshape(60,1)
outtest = np.append(yes, vee)
```

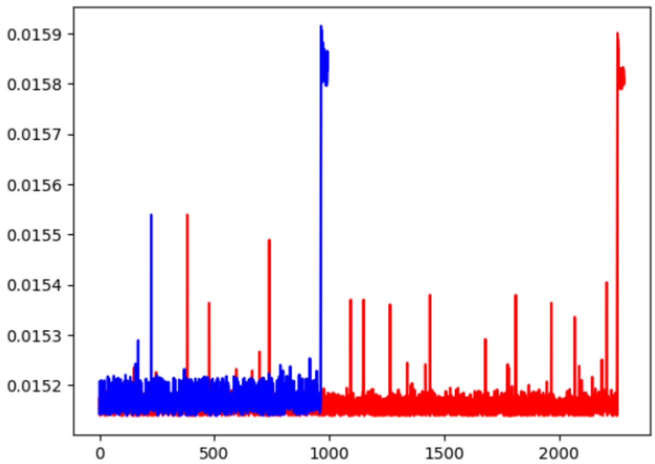
```
outtest.shape
```

```
(1986,)
```

```
df = pd.DataFrame(output)
df.to_csv('output2.csv', index=False)
testt = pd.DataFrame(outtest)
testt.to_csv('testout2.csv', index=False)
```

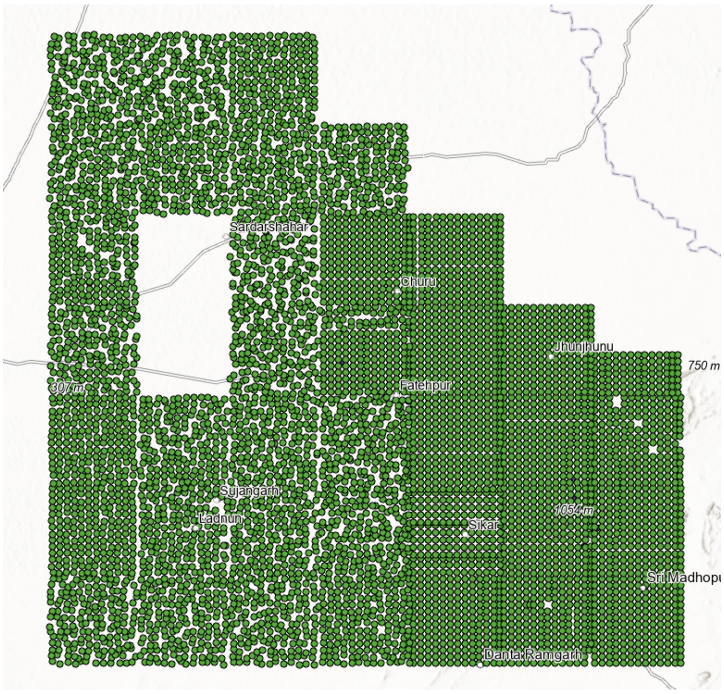
15. Visualizing data as a plot.

```
plt.plot(output, color = 'red', label = 'training')
plt.plot(outtest, color = 'blue', label = 'test')
#plt.title('')
#plt.xlabel('')
#plt.ylabel('')
#plt.legend()
plt.show()
```



IV. Creating Prospectivity Maps from LSTM's Predicted Cu deposit probability data.

The LSTM output Csv data containing Cu Deposit probability data was plotted as points with help of coordinate data in ArcGIS pro software.

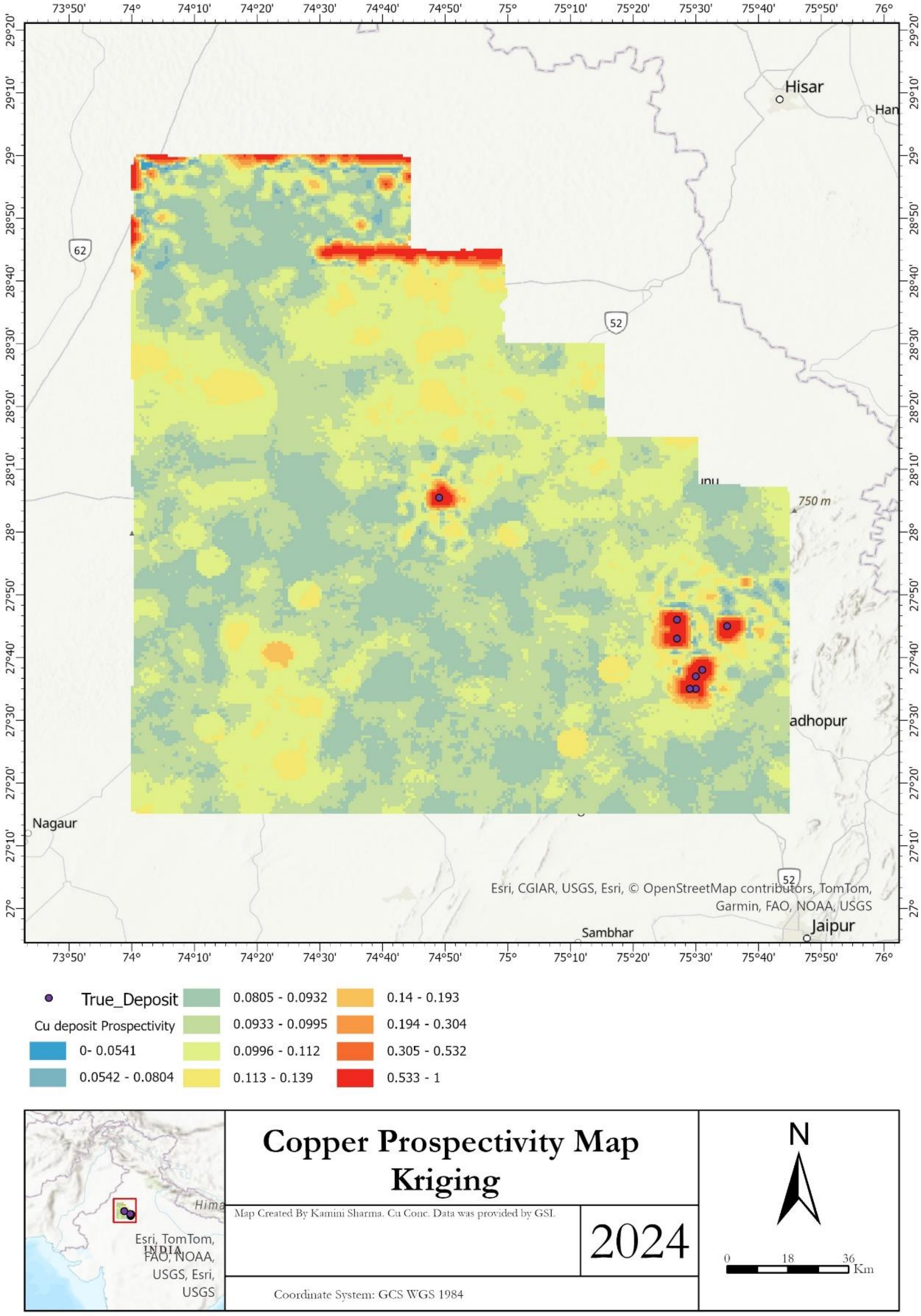


4. Results

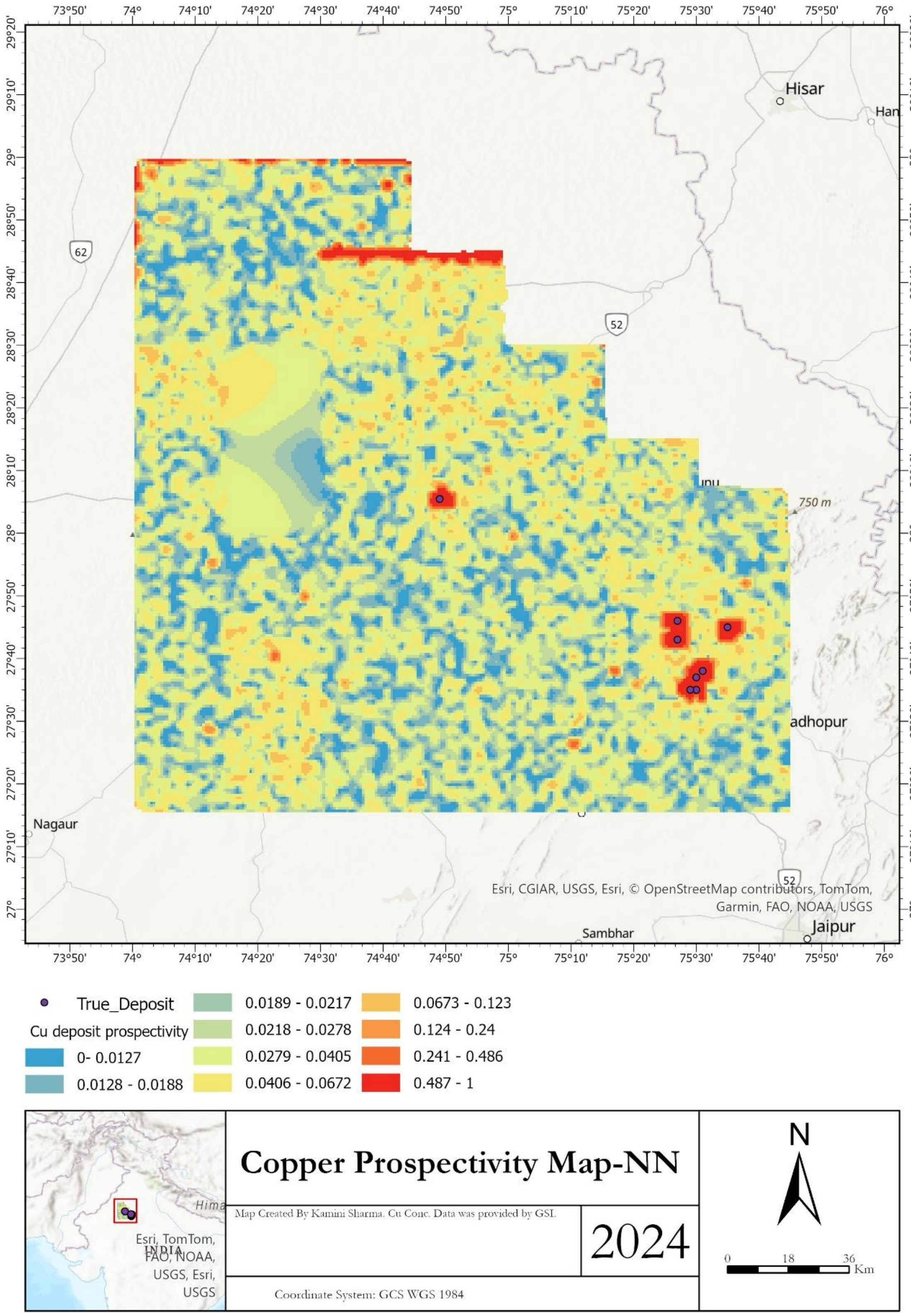
Using two techniques of interpolation- Natural Neighbor and Empirical Bayesian Kriging, Cu deposit probability values predicted by LSTM was converted to following Raster Maps showing probability of occurrence of a copper deposit at a place (Increasing from blue to red/ 0 to 1).Map also depicts true Ground deposits of copper which are found to be in High probability (red) zone.

The other high probability red areas are worth exploring by geologist for finding copper deposits. Our Map is limiting the prospective areas to a few km from 1000s. Such a map will reduce time and save money of Exploration geologist and companies searching for important and critical minerals.

1. Empirical Bayesian Kriging based Copper Prospectivity Map



2. Natural Neighbor – Copper Prospectivity Map



5. References

Binbin Li, Z. Y. (2023). One-dimensional convolutional neural network for mapping mineral prospectivity: A case study in Changba ore concentration area, Gansu Province. *Ore Geology Reviews*, 105573.