Appendix C

Download the Jupyter Notebook file called "RandomForests_FromCSV.ipynb" here:
https://github.com/charraden/Mineralogy-by-Random-Forests

Python package prerequisites (user to install): pandas, scikit-learn, joblib

This Jupyter notebook contains Python code to run the Random Forests method to estimate mineralogy from µXRF elemental data. Specifically, the sklearn package RandomForestRegressor was employed. The code first reads in table of µXRF X-ray counts with some rows (pixels) labelled with mineralogy. Using the elemental X-ray counts as inputs to the predictor model, the workflow allows the user to set the number of estimators (trees) and the random state. For this study, the number of estimators and random state were set at the scikit-learn default values of 100 and 42, respectively. The output of the workflow is a predicted label for each pixel in the X-ray counts data, as well as an accuracy score for each mineral based on the comparison of the labelled pixels predicted and labelled values. The output is a predicted label for each pixel, as well as an accuracy score for the prediction model based on the comparison of the labelled pixels predicted and labelled values. The tabular data can be viewed either in external software (e.g. QGIS, or ioGAS) or the user can add additional Python code to plot the tabular data as an image within the Python platform.

The code requires the elemental data comes in the same format as is exported from the code in Appendix A, with the element column headers containing the element abbreviation only. This code also requires an additional column called "MinLabel" which contains the mineral name for the labelled pixels. In our case, this was completed in QGIS by:

a) importing the wide format µXRF data as points (using the x and y values) making sure to bring the elemental data in as well,
b) adding and scaling the Axioscan composite image beneath the µXRF points,
c) selecting the points corresponding to a single mineral using the selection tool,
d) using the Field Calculator to add a new field called "MinLabel" and populating the with mineral name for the selected pixels,
e) repeating for all of the mineral labels in the map,
f) export the points attribute table as a .csv file.

Once the mineral labels have been added, the instructions below outline how to use the Random Forests Jupyter Notebook:

1) Save or copy the .csv file containing the mineral label and X-ray count data into the same folder as the Jupyter Notebook.
2) Add the file name in line 9, making sure to put it in quotes and adding the ".csv" file extension.

```
•[4]:
1  #Code developed by Cassady Harraden, Mineral Deposit Research Unit, University of British Columbia, June 2024
2  import pandas as pd
3  from sklearn.model_selection import train_test_split
4  from sklearn.ensemble import RandomForestClassifier
5  from sklearn.metrics import accuracy_score, classification_report
6  import joblib
7
8  #Load the CSV data
9  data = pd.read_csv('19FB-131_Counts_MinLabel.csv')
10
11 #Identify feature columns (excluding 'MinLabel', 'x', 'y', 'Video')
12 feature_columns = [col for col in data.columns if col not in ['MinLabel', 'x', 'y', 'Video']]
13
```

3) Make sure the target column name in lines 16, 22, and 23 is set to exactly the column name the mineral labels are stored in.

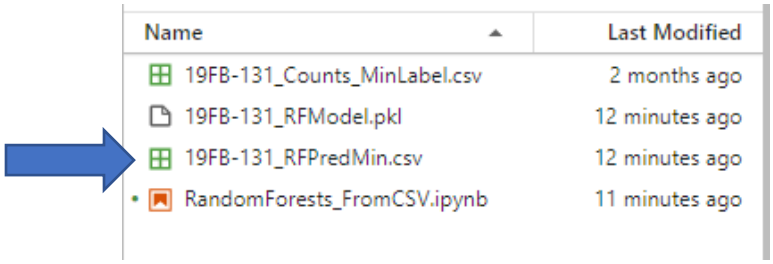
```
8  #Load the CSV data
9  data = pd.read_csv('19FB-131_Counts_MinLabel.csv')
10
11 #Identify feature columns (excluding 'MinLabel', 'x', 'y', 'Video')
12 feature_columns = [col for col in data.columns if col not in ['MinLabel', 'x', 'y', 'Video']]
13
14 # Prepare features and target
15 features = data[feature_columns]
16 target = data['MinLabel']
17
18 # Handle missing values in features
19 features = features.fillna(features.mean())
20
21 #Separate rows with no MinLabel
22 missing_minlabel_indices = data[data['MinLabel'].isna()].index
23 non_missing_minlabel_indices = data.dropna(subset=['MinLabel']).index
24
```

4) If you want to save the Random Forests model, give it a name (it will save the .pkl file) in line 48 (if not, comment this line of code out). Name the output .csv file in line 51.

```
36 #Predict for all rows
37 data['PredMin'] = rf_model.predict(features)
38
39 #Report model accuracy on MinLabel rows
40 y_pred_test = rf_model.predict(X_test)
41 accuracy = accuracy_score(y_test, y_pred_test)
42 report = classification_report(y_test, y_pred_test)
43 print(f'Model Accuracy: {accuracy}')
44 print('Classification Report:')
45 print(report)
46
47 # OPTIONAL: save the model for future use
48 joblib.dump(rf_model, '19FB-131_RFModel.pkl')
49
50 #Export Dataframe with predictions to a new CSV file
51 data.to_csv('19FB-131_RFPredMin.csv', index=False)
52
53 print("Model training, evaluation, and prediction completed successfully!")
54
```

5) Click inside the code box once, then hold down the SHIFT key, and hit ENTER on your keyboard to run the code.

6) When the code is finished, the .csv file containing the appended, predicted mineralogy will appear in the same folder as Jupyter Notebook. The model accuracy and statistics will also be reported beneath the code box.




```
Model Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

  Andalusite       1.00      1.00      1.00        12
    Corundum       1.00      1.00      1.00        20
   Muscovite       1.00      1.00      1.00        14
      Pyrite       1.00      1.00      1.00        25
      Quartz       1.00      1.00      1.00        14

    accuracy                           1.00        85
   macro avg       1.00      1.00      1.00        85
weighted avg       1.00      1.00      1.00        85

Model training, evaluation, and prediction completed successfully!
```