Supplementary Materials: Semi-automatization of support vector machines to map lithium (Li) bearing pegmatites

Joana Cardoso-Fernandes*[1,2], Ana C. Teodoro[1,2], Alexandre Lima[1,2] and Encarnación Roda-Robles[3]

[1] Department of Geosciences, Environment and Land Planning, Faculty of Sciences, University of Porto, Rua Campo Alegre, 4169-007 Porto, Portugal; amteodor@fc.up.pt (A.C.T.); allima@fc.up.pt (A.L.)
[2] Institute of Earth Sciences (ICT), Pole of University of Porto, 4169-007 Porto, Portugal
[3] Departamento Mineralogía y Petrología, Univ. País Vasco (UPV/EHU), Barrio Sarriena, 48940 Leioa, Bilbao, Spain; encar.roda@ehu.es
* Correspondence: joana.fernandes@fc.up.pt

1. Splitting the data into training and test subsets

To ensure the independence between the training and test subsets, the samples were randomly divided according to their respective region of interest (ROI). This means that the samples belonging to the same ROI will either belong to the training or the test set. Unlike the scikit-learn's `train_test_split` function, the `GroupShuffleSplit` iterator does not allow to stratify the sample splitting, i.e., `GroupShuffleSplit` does not guarantee that all classes will be present in both subsets. Therefore, it is necessary to find, by trial and error method, a random state seed that satisfies this condition (in this work the random seed found was 1020).

The data for this example can be found here: https://drive.google.com/drive/folders/1K43om-5XMh0DBwSqwDF-Tz4CEW2ul92a?usp=sharing.

The Sentinel-2 image (S2B_MSIL2A_20190907T112119_N0213_R037_T29TPF_20190907T144322) can be downloaded here: https://scihub.copernicus.eu/dhus/#/home.

```
## The scikit-learn version is 0.20.1
# Start with the general imports
import numpy as np
import pandas as pd

# Load the data
path = '/user/…/'
S2_dataset = pd.read_csv(path, sep='\t')

# Load the ROI information of every sample
path2 = '/user/…/'
groups = pd.read_csv(path2, sep='\t')
groups = groups['Groups'] # remove second column
groups = np.array(groups)

# Create features
X_S2 = S2_dataset.drop('Classname', axis=1)
X_S2 = np.array(X_S2)
print("Features shape: {}".format(X_S2.shape))
```

```
# Create target
y_S2 = S2_dataset['Classname']
y_S2= np.array(y_S2)
print("Target shape: {}".format(y_S2.shape))
# Use GroupShuffleSplit to create train/test set
from sklearn.model_selection import GroupShuffleSplit

train_inds, test_inds = next(GroupShuffleSplit(test_size=0.25,
     random_state=1020).split(X_S2, y_S2, groups))

X_train, X_test, y_train, y_test = X_S2[train_inds],
     X_S2[test_inds], y_S2[train_inds], y_S2[test_inds]

print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

2. First stage grid-search

Having the data loaded and the training/test subsets created, it is possible to instantiate the grid-search process. Since the first stage requires more control from the operator, each model was addressed separately. In practice, the first stage grid-search was repeated every time one of the following variables changed: (i) type of data (imbalanced/balanced); (ii) type of model (Linear model, linear kernel, polynomial kernel, and RBF kernel); (iii) parameter range.

As mentioned, the metric score employed in this first stage was the harmonic mean of the precision and recall, i.e., the F1-score of the selected classes (Li-bearing pegmatite and Metasediments). For that, it was necessary to create a personalized metric score to pass was one of the `GridSearchCV` attributes. Instead of using the `f1_score` function directly, we decided to use the `fbeta_score` since it gives the operator the ability to attribute a bigger weight to precision or to recall according to the final goal.

For demonstration purposes, only the source code employed for the Linear model using the class-weight balancing strategy is shown.

```
# General imports
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import fbeta_score
from sklearn.metrics import make_scorer

# Make notebook stable across runs
np.random.seed(42)
```

```
# Make a personalized score (F1-score), where precision and recall
have the same weight
score_beta = make_scorer(fbeta_score, beta=1,
     labels = ['Li-bearing pegmatite','Metasediments'],
     average = 'micro')

# Specify the parameter range (search space) using a dictionary
    # using the parameters used by Noi & Kappas 2018
param_grid = {'C':[0.25, 0.5, 1, 2, 4, 8, 16, 32, 64, 128]}
print("Parameter grid:\n{}".format(param_grid))

# Create grid-search
grid_search = GridSearchCV(LinearSVC(class_weight='balanced'),
     param_grid, cv=5, n_jobs=-1, verbose=1, scoring=score_beta,
     return_train_score=True)

# Fit grid-search
grid_search.fit(X_train, y_train)

# View best hyperparameters
print('Best C:',  grid_search.best_estimator_.get_params()['C'])

# Show best cross-validation F1-score
print("Best CV F1-score: {:.6f}".format(grid_search.best_score_))

# Show the results of the grid search
results = pd.DataFrame(grid_search.cv_results_)
results

# Show results in an image
scores = np.array(results.mean_test_score)
C_param = [0.25, 0.5, 1, 2, 4, 8, 16, 32, 64, 128]
plt.plot(C_param, scores)
plt.xlabel('C')
plt.ylabel('scores')
```

3. Second stage grid-search, model evaluation and image prediction

This section corresponds to the automatized part of the image classification process. During the second stage grid-search, the Linear model, linear kernel and RBF kernel are confronted, and the best model is automatically returned. The best model is evaluated using the test subset and used to predict the whole image.

The following source code exemplifies the automated process of model selection, evaluation and classification for the imbalanced dataset. As before, the training/test subsets are already loaded.

```python
# General imports
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import pandas as pd
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
import rasterio as rio
from rasterio import plot
import geopandas as gpd


# Make notebook stable accross runs
np.random.seed(42)

# Create a pipeline
pipe = Pipeline([("classifier", SVC())])

# Create dictionary with candidate algorithms and their parameters
search_space = [{"classifier": [LinearSVC()],
    "classifier__max_iter": [2500],
    "classifier__C": [30.5, 31, 31.5, 32, 32.5, 33, 33.5]},
    {"classifier": [SVC()],
     "classifier__kernel":['linear'],
    "classifier__C": [6.5, 7, 7.5, 8, 8.5, 9, 9.5]},
    {"classifier": [SVC()],
     "classifier__kernel":['rbf'],
     "classifier__C": [0.5, 1, 1.5, 2, 2.5, 3, 3.5],
     "classifier__gamma": [1.4,1.6,1.8,2,2.2,2.4,2.6]}]

# Create grid-search
grid_search = GridSearchCV(pipe, search_space, cv=5, n_jobs=-1,
    verbose=1, scoring= 'accuracy', return_train_score=True)

# Fit grid-search
best_model = grid_search.fit(X_train, y_train)

print("Best parameters: {}".format(best_model.best_params_))
print("Best CV score: {:.6f}".format(best_model.best_score_))
print("Test-score: {:.6f}".format(best_model.score(X_test,y_test)))
```

```
# Show the results of the grid search
results = pd.DataFrame(best_model.cv_results_)

# Nested cross-validation to evaluate the best model
from sklearn.model_selection import cross_val_score
scores= cross_val_score(grid_search, X_S2, y_S2)
print("Cross-validation scores: ", scores)
print("Mean cross-validation score: ", scores.mean())


### Evaluate the classifier predictions
# Predict labels for new data
y_predicted = best_model.predict(X_test)

# Get classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predicted))

# Show confusion matrix
from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(y_test, y_predicted)
class_names = np.unique(S2_dataset["Classname"])
dataframe = pd.DataFrame(matrix, index=class_names,
    columns=class_names)

plt.figure(figsize = (16,10))
sns.heatmap(dataframe, annot=True, cbar=None,
    fmt='g',cmap='YlGnBu', annot_kws={"size": 15})
plt.title("Confusion Matrix",fontsize=32, pad=15),
    plt.tight_layout()
plt.ylabel("True Class",fontsize=25, labelpad=15),
    plt.xlabel("Predicted Class",fontsize=25, labelpad=15)
plt.xticks(fontsize=16,rotation = 45)
plt.yticks(fontsize=16)
plt.show()

# Compute kappa statistics
from sklearn.metrics import cohen_kappa_score
kappa = cohen_kappa_score(y_test, y_predicted)
print("kappa statistics: ", kappa)
```

```
## Predict the whole image
#Load the Sentinel-2 bands
b2 = rio.open('/user/…/B2.tif')
b3 = rio.open('/user/…/B3.tif')
b4 = rio.open('/user/…/B4.tif')
b8 = rio.open('/user/…/B8.tif')
b11 = rio.open('/user/…/B11.tif')
b12 = rio.open('/user/…/B12.tif')

B2 = b2.read(1).astype('float32')
B3 = b3.read(1).astype('float32')
B4 = b4.read(1).astype('float32')
B8 = b8.read(1).astype('float32')
B11 = b11.read(1).astype('float32')
B12 = b12.read(1).astype('float32')

# Write image to stack
reshaped_img = np.dstack([B2, B3, B4, B8, B11, B12])
print(reshaped_img.shape)

# Predict image
class_prediction = best_model.predict(reshaped_img.reshape(-1, 6))

# Reshape image back into a 2D matrix
class_prediction =
    class_prediction.reshape(reshaped_img[:, :, 0].shape)

# Convert the classes' string labels to a numpy array
class_prediction[class_prediction == 'Agricultural fields'] = 0
class_prediction[class_prediction == 'Burned areas'] = 1
class_prediction[class_prediction == 'Granite'] = 2
class_prediction[class_prediction == 'Li-bearing pegmatite'] = 3
class_prediction[class_prediction == 'Metasediments'] = 4

class_prediction = class_prediction.astype(float)
```

```python
# Export final image
Imb_acc = rio.open('/user/…/Final_classified_map.tiff','w',
                          driver='Gtiff',
                          width = b4.width,
                          height = b4.height,
                          count=1,
                          crs = b4.crs,
                          transform = b4.transform,
                          dtype='float64')
Imb_acc.write(class_prediction,1)
Imb_acc.close()
```