part_3_segementation_of_green_var

January 24, 2025

1 Introduction

In this part of the analysis, we focused on reassigning individuals into the identified clusters using illustrative variables. Building on the clustering results from the first step, we evaluated two algorithms, Random Forest and Gradient Boosting. Here, we focus on the segmentation of the green variables. In the first part, using the orange variables and then using a specific set of variables.

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.decomposition import PCA
from sklearn.utils.class_weight import compute_class_weight
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

1.1 Data Loading

[2]:
```
file_path = "fic_epita_kantar_codes.csv"
data = pd.read_csv(file_path, sep=';')

# Define Orange and Green variable groups
orange_vars = [
    "A9_1_slice", "A9_2_slice", "A9_3_slice", "A9_4_slice", "A9_5_slice",
    "A9_6_slice", "A9_7_slice", "A9_8_slice", "A9_9_slice", "A9_10_slice",
    "A9_11_slice", "A9_12_slice", "A9_13_slice", "A9_14_slice", "A9_15_slice",
    "A9_16_slice", "A10_1_slice", "A10_2_slice", "A10_3_slice", "A10_4_slice",
    "A10_5_slice", "A10_6_slice", "A10_7_slice", "A10_8_slice",
    "A11_1_slice", "A11_2_slice", "A11_3_slice", "A11_4_slice", "A11_5_slice",
    "A11_6_slice", "A11_7_slice", "A11_8_slice", "A11_9_slice", "A11_10_slice",
    "A11_11_slice", "A11_12_slice", "A11_13_slice"
]
green_vars = [
    "A11_1_slice", "A12", "A13", "A14", "A4", "A5", "A5bis", "A8_1_slice",
```

```
    "A8_2_slice", "A8_3_slice", "A8_4_slice", "B1_1_slice", "B1_2_slice",
    "B2_1_slice", "B2_2_slice", "B3", "B4", "B6", "C1_1_slice", "C1_2_slice",
    "C1_3_slice", "C1_4_slice", "C1_5_slice", "C1_6_slice", "C1_7_slice",
    "C1_8_slice", "C1_9_slice"
]
specific_vars = [
    "rs3", "rs5", "rs6", "RS1", "RS191", "RS192", "RS193", "RS102RECAP",
    "rs11recap2", "RS11recap", "RS193bis", "RS2Recap", "RS56Recap",
    "RS2", "RS11", "RS102"
]
```

1.2 Adding the cluster column using the best algo with the best number of cluster found in part 1

[3]:
```
# Fill missing values in Green variables
green_data = data[green_vars].fillna(0)

# Scale the data
scaler = StandardScaler()
scaled_green_data = scaler.fit_transform(green_data)

# Perform KMeans clustering with 4 clusters
kmeans = KMeans(n_clusters=4, random_state=42)
data['cluster_green'] = kmeans.fit_predict(scaled_green_data)
```

1.3 Segmentation based on green variable

[4]:
```
# Extract features (Green variables) and target (Orange clusters)
X_orange = data[orange_vars].fillna(0)
y_green = data['cluster_green']

# Split data into training and test sets
X_train_orange, X_test_orange, y_train, y_test = train_test_split(
    X_orange, y_green, test_size=0.2, random_state=42
)

# Train and Evaluate model
rf_orange = RandomForestClassifier(random_state=42)
rf_orange.fit(X_train_orange, y_train)

y_pred_orange = rf_orange.predict(X_test_orange)
print("Classification Report (Green Variables) with Random Forest:\n",
  classification_report(y_test, y_pred_orange))
```

```
Classification Report (Green Variables) with Random Forest:
               precision    recall  f1-score   support
```

```
           0       0.30      0.07      0.11       103
           1       0.56      0.28      0.37       256
           2       0.48      0.40      0.44       100
           3       0.60      0.85      0.71       541

    accuracy                           0.58      1000
   macro avg       0.49      0.40      0.41      1000
weighted avg       0.55      0.58      0.53      1000
```

[5]:

```python
# Extract features (Orange variables) and target (Green clusters)
X2 = data[orange_vars].fillna(0)
y2 =  data['cluster_green']

# Split data into training and test sets
X_train2, X_test2, y_train2, y_test2 = train_test_split(
    X2, y2, test_size=0.2, random_state=42
)

# Train and Evaluate model
gb = GradientBoostingClassifier(random_state=42)
gb.fit(X_train2, y_train2)
y_pred2 = gb.predict(X_test2)
print("Classification Report (Specific Variables with Gradient Boosting):\n",
 ↪classification_report(y_test2, y_pred2))
```

```
Classification Report (Specific Variables with Gradient Boosting):
               precision    recall  f1-score   support

           0       0.44      0.18      0.26       103
           1       0.60      0.30      0.40       256
           2       0.47      0.40      0.43       100
           3       0.61      0.84      0.70       541

    accuracy                           0.59      1000
   macro avg       0.53      0.43      0.45      1000
weighted avg       0.58      0.59      0.55      1000
```

1.4 Optimizing results to try to improve the average performance we get

[6]:

```python
# Extract features (Green variables) and target (Orange clusters)
X_orange = data[orange_vars].fillna(0)
y_green = data['cluster_green']

# Split data into training and test sets
X_train_orange, X_test_orange, y_train, y_test = train_test_split(
    X_orange, y_green, test_size=0.2, random_state=42
```

```python
)

# Define parameter grid for Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize a Random Forest classifier
rf = RandomForestClassifier(random_state=42)

# Perform GridSearchCV to optimize hyperparameters
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           scoring='accuracy', cv=3, verbose=0, n_jobs=-1)

# Fit the GridSearchCV on training data
grid_search.fit(X_train_orange, y_train)

# Get the best parameters and score
best_params = grid_search.best_params_
best_score = grid_search.best_score_
print(f"Best Hyperparameters: {best_params}")
print(f"Best Cross-Validation Accuracy: {best_score:.4f}")

# Train and Evaluate model with optimized hyperparameters
rf_optimized = RandomForestClassifier(**best_params, random_state=42)
rf_optimized.fit(X_train_orange, y_train)
y_pred_orange_optimized = rf_optimized.predict(X_test_orange)

# Print the classification report
print("Classification Report (Green Variables) with Random Forest Optimization:
 ↪\n",
      classification_report(y_test, y_pred_orange_optimized))
```

```
Best Hyperparameters: {'bootstrap': True, 'max_depth': 30, 'min_samples_leaf':
2, 'min_samples_split': 5, 'n_estimators': 50}
Best Cross-Validation Accuracy: 0.6073
Classification Report (Green Variables) with Random Forest Optimization:
              precision    recall  f1-score   support

           0       0.58      0.11      0.18       103
           1       0.61      0.26      0.37       256
           2       0.42      0.33      0.37       100
           3       0.60      0.87      0.71       541
```

```
    accuracy                           0.58      1000
   macro avg       0.55      0.39      0.41      1000
weighted avg       0.58      0.58      0.53      1000
```

[7]:
```
# Extract features (Orange variables) and target (Green clusters)
X2 = data[orange_vars].fillna(0)
y2 = data['cluster_green']

# Split data into training and test sets
X_train2, X_test2, y_train2, y_test2 = train_test_split(
    X2, y2, test_size=0.2, random_state=42
)

# Define parameter grid for Gradient Boosting
param_grid2 = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize a Gradient Boosting classifier
gb2 = GradientBoostingClassifier(random_state=42)

# Perform GridSearchCV to optimize hyperparameters
grid_search2 = GridSearchCV(estimator=gb2, param_grid=param_grid2,
                            scoring='accuracy', cv=3, verbose=0, n_jobs=-1)

# Fit the GridSearchCV on training data
grid_search2.fit(X_train2, y_train2)

# Get the best parameters and score
best_params2 = grid_search2.best_params_
best_score2 = grid_search2.best_score_
print(f"Best Hyperparameters (Gradient Boosting): {best_params2}")
print(f"Best Cross-Validation Accuracy (Gradient Boosting): {best_score2:.4f}")

# Train and Evaluate model with optimized hyperparameters
gb_optimized = GradientBoostingClassifier(**best_params2, random_state=42)
gb_optimized.fit(X_train2, y_train2)
y_pred2_optimized = gb_optimized.predict(X_test2)

# Print the classification report
print("Classification Report (Specific Variables with Gradient Boosting␣
 ↪Optimization):\n",
```

```
    classification_report(y_test2, y_pred2_optimized))
```

```
Best Hyperparameters (Gradient Boosting): {'learning_rate': 0.1, 'max_depth': 3,
'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 100}
Best Cross-Validation Accuracy (Gradient Boosting): 0.6070
Classification Report (Specific Variables with Gradient Boosting Optimization):
              precision    recall  f1-score   support

           0       0.41      0.17      0.24       103
           1       0.61      0.31      0.41       256
           2       0.44      0.35      0.39       100
           3       0.61      0.84      0.71       541

    accuracy                           0.59      1000
   macro avg       0.52      0.42      0.44      1000
weighted avg       0.57      0.59      0.55      1000
```

1.5 Segmentation based on specific variable

[8]:
```
# For Random Forest
# Extract features (Specific variables) and target (Green clusters)
X_specific = data[specific_vars].fillna(0)
y_green = data['cluster_green']

# Split data into training and test sets
X_train_specific, X_test_specific, y_train, y_test = train_test_split(
    X_specific, y_green, test_size=0.2, random_state=42
)

# Compute class weights
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train),
    y=y_train
)

# Convert weights into a dictionary
class_weights_dict = {i: weight for i, weight in enumerate(class_weights)}

# Train and Evaluate model
rf_specific = RandomForestClassifier(random_state=42,␣
 ↪class_weight=class_weights_dict)
rf_specific.fit(X_train_specific, y_train)
y_pred_specific = rf_specific.predict(X_test_specific)
print("Classification Report (Specific Variables) with Random Forest:\n",␣
 ↪classification_report(y_test, y_pred_specific))
```

```
Classification Report (Specific Variables) with Random Forest:
               precision    recall  f1-score   support

           0       0.22      0.34      0.27       103
           1       0.64      0.67      0.65       256
           2       0.18      0.10      0.13       100
           3       0.67      0.64      0.66       541

    accuracy                           0.56      1000
   macro avg       0.43      0.44      0.43      1000
weighted avg       0.57      0.56      0.56      1000
```

[9]:
```
# For Gradient Boosting
# Extract features (Specific variables) and target (Green clusters)
X_specific = data[specific_vars].fillna(0)

# Split data into training and test sets
X_train_specific, X_test_specific, y_train, y_test = train_test_split(
    X_specific, y_green, test_size=0.2, random_state=42
)

# Train and Evaluate model
gb_specific = GradientBoostingClassifier(random_state=42)
gb_specific.fit(X_train_specific, y_train)
y_pred_specific = gb_specific.predict(X_test_specific)
print("Classification Report (Specific Variables) with Gradient Boosting:\n",␣
 ↪classification_report(y_test, y_pred_specific))
```

```
Classification Report (Specific Variables) with Gradient Boosting:
               precision    recall  f1-score   support

           0       0.14      0.01      0.02       103
           1       0.74      0.69      0.71       256
           2       0.10      0.01      0.02       100
           3       0.67      0.92      0.77       541

    accuracy                           0.68      1000
   macro avg       0.41      0.41      0.38      1000
weighted avg       0.58      0.68      0.61      1000
```

1.6 Optimizing results to try to improve the average performance we get

[10]:
```
# For Random Forest
# Extract features (Specific variables) and target (Green clusters)
X_specific = data[specific_vars].fillna(0)
```

```
# Split data into training and test sets
X_train_specific, X_test_specific, y_train, y_test = train_test_split(
    X_specific, y_green, test_size=0.2, random_state=42
)

# Define parameter grid for Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize a Random Forest classifier
rf = RandomForestClassifier(random_state=42)

# Perform GridSearchCV to optimize hyperparameters
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           scoring='accuracy', cv=3, verbose=0, n_jobs=-1)

# Fit the GridSearchCV on training data
grid_search.fit(X_train_specific, y_train)

# Get the best parameters and score
best_params = grid_search.best_params_
best_score = grid_search.best_score_
print(f"Best Hyperparameters: {best_params}")
print(f"Best Cross-Validation Accuracy: {best_score:.4f}")

# Train and Evaluate model with optimized hyperparameters
rf_optimized = RandomForestClassifier(**best_params, random_state=42)
rf_optimized.fit(X_train_specific, y_train)
y_pred_specific_optimized = rf_optimized.predict(X_test_specific)
print("Classification Report (Specific Variables with Hyperparameter␣
 ↪Optimization):\n",
      classification_report(y_test, y_pred_specific_optimized))
```

```
Best Hyperparameters: {'bootstrap': True, 'max_depth': 20, 'min_samples_leaf':
4, 'min_samples_split': 10, 'n_estimators': 200}
Best Cross-Validation Accuracy: 0.6718
Classification Report (Specific Variables with Hyperparameter Optimization):
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       103
           1       0.71      0.71      0.71       256
           2       0.00      0.00      0.00       100
           3       0.68      0.93      0.78       541
```

```
    accuracy                           0.68      1000
   macro avg       0.35      0.41      0.37      1000
weighted avg       0.55      0.68      0.60      1000

/home/floflo/Documents/epita/epita-ml-scia/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/floflo/Documents/epita/epita-ml-scia/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/floflo/Documents/epita/epita-ml-scia/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

[11]:

```python
# For Gradient Boosting
# Define parameter grid for Gradient Boosting
param_grid2 = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize a Gradient Boosting classifier
gb2 = GradientBoostingClassifier(random_state=42)

# Perform GridSearchCV to optimize hyperparameters
grid_search2 = GridSearchCV(estimator=gb2, param_grid=param_grid2,
                            scoring='accuracy', cv=3, verbose=0, n_jobs=-1)

# Fit the GridSearchCV on the training data (using y_train instead of y_green)
grid_search2.fit(X_train_specific, y_train)

# Get the best parameters and score
best_params2 = grid_search2.best_params_
best_score2 = grid_search2.best_score_
print(f"Best Hyperparameters (Gradient Boosting): {best_params2}")
print(f"Best Cross-Validation Accuracy (Gradient Boosting): {best_score2:.4f}")
```

```python
# Train and Evaluate model with optimized hyperparameters
gb_optimized = GradientBoostingClassifier(**best_params2, random_state=42)
gb_optimized.fit(X_train_specific, y_train)
y_pred_specific_optimized = gb_optimized.predict(X_test_specific)
print("Classification Report (Specific Variables with Gradient Boosting␣
 ↪Optimization):\n",
      classification_report(y_test, y_pred_specific_optimized))
```

```
Best Hyperparameters (Gradient Boosting): {'learning_rate': 0.01, 'max_depth':
3, 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 100}
Best Cross-Validation Accuracy (Gradient Boosting): 0.6788
Classification Report (Specific Variables with Gradient Boosting Optimization):
               precision    recall  f1-score   support

           0       0.00      0.00      0.00       103
           1       0.75      0.69      0.72       256
           2       0.00      0.00      0.00       100
           3       0.67      0.94      0.78       541

    accuracy                           0.69      1000
   macro avg       0.35      0.41      0.37      1000
weighted avg       0.55      0.69      0.61      1000


/home/floflo/Documents/epita/epita-ml-scia/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/floflo/Documents/epita/epita-ml-scia/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/floflo/Documents/epita/epita-ml-scia/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

1.6.1 Conclusion

In this part of the analysis, we reassigned individuals to their respective clusters using the illustrative variables derived from the Green segmentation. The segmentation yielded pretty average performance when using either the Green variables or the specific given variables. Optimizing hyperparameters did not significantly improve the results, with accuracies of approximately 0.6 for the Green variables and 0.7 for the specific given variables.

Both Random Forest and Gradient Boosting produced similar results, with a difference of only 0.01

in accuracy in each case. Given this negligible difference and the fact that Random Forest is a faster algorithm, it is the recommended choice, as the slight performance improvement offered by Gradient Boosting does not justify the additional computational cost.