Machine Learning and Data Mining Project - Flag Study

Mert Akpinar
Email: mert.akpinar@studio.unibo.it

***Abstract*—This machine-learning project explores various techniques and models for the classification task on the Flags dataset sourced from https://archive.ics.uci.edu/dataset/40/flags. The primary objective is to predict the religious affiliation of countries based on their respective flag attributes.**

1. Introduction

National symbols frequently incorporate religious elements, and this is evident in the national anthems and flags of many countries. The objective of this undertaking is to categorize national flags based on their distinctive features. It is widely acknowledged that the colors and symbols chosen for a national flag are deliberate, each carrying specific meanings. These colors may be proudly donned by the citizens to express their patriotism, and alterations to a flag's design might occur in response to significant historical events. The project attempts to forecast a country's religious affiliation based on the characteristics of its flag.

1.1. Dataset

Dataset The dataset encompasses diverse information about countries and their flags. Specifically, it comprises data on 194 countries, with 30 variables or attributes for each country. To enhance organization, I have partitioned the dataset into three tables based on the type of variables. Table 1, labeled Geographical characteristics, encompasses variables associated with the geographical aspects of the flags. Meanwhile, Table 2, named Colour Characteristics, includes variables pertaining to the color attributes of the flags.

Variable Name	Variable Type	Description
name	Character	Name of the country concerned
landmass	Factor (6 levels)	Geographical Continent
zone	Factor (4 levels)	Geographic quadrant, based on Greenwich and the Equator
area	Numerical	Area in thousands of square km
population	Numerical	Population in round millions
language	Factor (8 levels)	Language
religion	Factor (5 levels)	Religion

Table 1 - Geographical Characteristics

2. Classification

In this part of the project, I tried to categorize the nation flags and identify their religion according to the characteristics of the flag. At the beginning of the analysis,

Variable Name	Variable Type	Description
colours	Numerical	Colours Number of different colours in the flag
red	Factor (2 levels)	0 if red absent, 1 if red present in the flag
green	Factor (2 levels)	0 if green absent, 1 if green present in the flag
blue	Factor (2 levels)	0 if blue absent, 1 if blue present in the flag
gold	Factor (2 levels)	0 if gold absent, 1 if gold present in the flag
white	Factor (2 levels)	0 if white absent, 1 if white present in the flag
black	Factor (2 levels)	0 if black absent, 1 if black present in the flag
orange	Factor (2 levels)	0 if orange absent, 1 if orange present in the flag
mainhue	Factor (8 levels)	Predominant colour in the flag
topleft	Factor (8 levels)	Colour in the top-left corner (moving right to decide tie-breaks)
topright	Factor (8 levels)	Colour in the bottom-left corner (moving left to decide tie-breaks)

Table 2 – Colour Characteristics

Variable Name	Variable Type	Description
bars	Numerical	Number of vertical bars in the flag
stripes	Numerical	Number of horizontal stripes in the flag
circles	Numerical	Number of circles in the flag
crosses	Numerical	Number of (upright) crosses
saltires	Numerical	Number of diagonal crosses
quarters	Numerical	Number of quartered sections
sunstars	Numerical	Number of sun or star symbols
crescent	Numerical	1 if a crescent moon symbol present, else 0
triangle	Numerical	1 if any triangles present, 0 otherwise
icon	Factor (2 levels)	1 if an inanimate image present (e.g., a boat), otherwise 0
animate	Factor (2 levels)	1 if an animate image (e.g., an eagle, a tree, a human hand) present, 0 otherwise
text	Factor (2 levels)	1 if any letters or writing on the flag (e.g., a motto or slogan), 0 otherwise

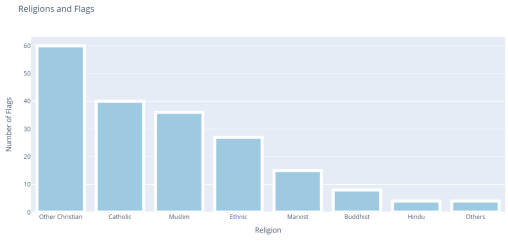
Table 3 - Geometrical Characteristics

I did a descriptive analysis in order to understand better the dataset. Then, I applied various classification classifiers for comparison in accuracy. The classification analysis includes the following techniques:

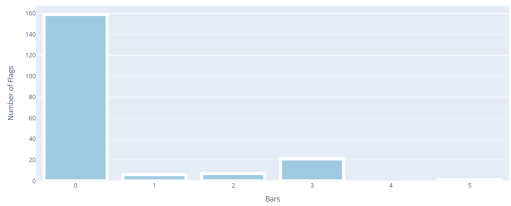
- Decision Trees
- Random Forest
- Support Vector Machines

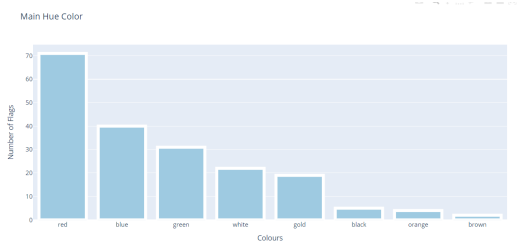
2.1. Descriptive Analysis

After importing the dataset into Google Colab, I visualized some of the characteristics of the dataset to have a better understanding.



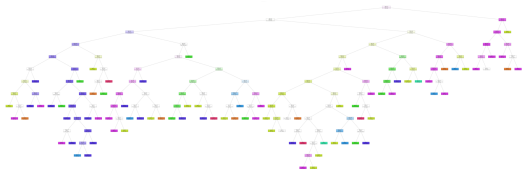
From these 3 figures, we can see that the most widespread religions are Catholicism and Christianity, most of the countries have 3 different colours in their flags and the most popular color is red.





2.2. Decision Tree

As a first approach, I tried to solve the classification problem with a Decision Tree Classifier. After splitting the dataset to train and test subsets with a 0.8:0.2 ratio and training it, the classifier resulted in 33.3% accuracy. The model after decision tree classification is very complicated because there are lots of mixed variables; categorical and numerical.



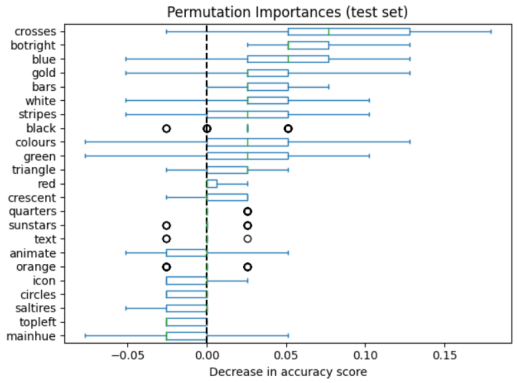
As can be seen from the figure above, the decision tree has an enormous number of rules. The fact that the dataset contains a tiny number of elements makes the learning phase of the model difficult. That's why for model evaluation I used K-fold cross-validation. I chose this method because the dataset has only 194 observations, which is insufficient for a simple split for training and test datasets. After trying different values for the k constant on folding, the optimal k-value was 5 and it helped the model to achieve 40% accuracy.

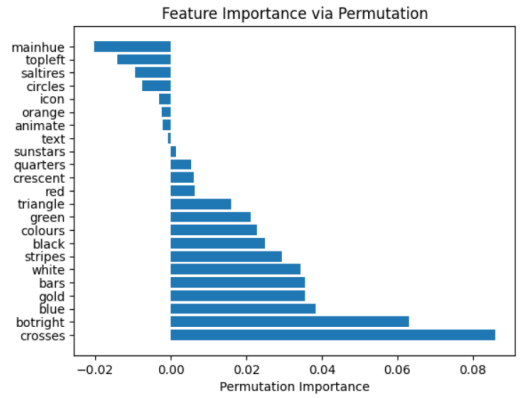
Having these many rules in the decision tree makes the depth of the tree greater. It may also make the learning phase harder by reducing the generalization rules for elements. If we consider the size of the dataset, having a large number of rules is expected since the religions for the flags are not spread equally in numbers. I wanted to discover the optimal tree depth for better accuracy for the tree model. To get the best hyperparameter I used Grid Search. I passed depth values from 1 to the current depth of the decision tree and in return, the optimal depth of the tree with the highest accuracy was 11. After getting the optimal parameter from grid search, I have obtained the tree shown in the figure below with 43.3% accuracy.

2.3. Random Forest

After tuning the parameters for the decision tree model and obtaining greater accuracy, I wanted to have a more powerful model that could understand the data better. Because with this dataset the decision tree tends to create a complex model that fits the training data very closely, which leads to overfitting. The next step for me was to create a Random Forest Classifier to acquire multiple decision trees and the learning phase that comes from the combination of these various trees. By aggregating the outputs of multiple trees, the Random Forest model tends to generalize better to new data. After testing various values for the number of subtrees, I achieved a higher accuracy of 51.3% with 50 subtrees.

Furthermore, I wanted to estimate the importance of different features in the model to assess how much the model depends on each feature for making accurate predictions. Since there are very few samples and 23 features for evaluation of the rules, I thought it is reasonable to select the features that play a bigger role in prediction. Even though Random Forest Classifier comes with the impurity-based feature importance, it suffers from being computed on statistics derived from the training dataset, therefore it does not reflect the ability of the feature to be useful to make predictions that generalize to the test set. Instead, I chose to use the Permutation Feature Importance method, which involves shuffling the values of a single feature while keeping other features unchanged and measuring the impact on the model's performance. Features with a greater decrease in performance after permutation are considered more important.





After obtaining the permutation importance chart, I selected the first 15 features and fit the Random Forest model again. After the training, the model resulted with 56.4% accuracy. By using the permutation importance technique and setting the "min_samples_split" feature for the classifier to 12, the model's score increased, and also overfitting was prevented.

If we consider previous training and test accuracy scores, there was a huge difference between them. That means the model learns the training set so closely but fails to predict future observations. By determining the minimum number of samples required to split an internal node, the number of rules and depth of subtrees are reduced and the rules created by the model became more consistent for both training and test sets. We can see the convergence of training and test scores.

```
# Retrieve important columns
selected_features = []
for i in range(len(sorted_perm_importance)):
  if sorted_perm_importance.Importance[i] > 0:
    selected_features.append(sorted_perm_importance.Feature[i])

X_train_selected = X_train[selected_features]
X_test_selected = X_test[selected_features]

rf_selected = RandomForestClassifier(n_estimators=50, random_state=42, min_samples_split=12)
rf_selected.fit(X_train_selected, y_train)

print('Training accuracy: ', np.mean(rf_selected.predict(X_train_selected) == y_train)*100)
print('Test accuracy: ', np.mean(rf_selected.predict(X_test_selected) == y_test)*100)
```

```
Training accuracy:  69.03225806451613
Test accuracy:  56.41025641025641
```

2.4. Support Vector Machine

With Support Vector Machines (SVM), we are looking for the optimal separating hyperplane between the two classes by maximizing the margin between the classes' closest points. Firstly, I have searched for the optimal cost and gamma model parameters by supplying parameter ranges. After comparing the results with different values for these two parameters, I ran the model and calculated the accuracy with optimal parameter values for gamma and the cost. With the help of feature selection and values of gamma and cost variables, I achieved 61.5% accuracy with the Support Vector Machine model.

2.5. Gradient Boosting

After applying bagging models and various hyperparameter techniques, I wanted to test the performance of a boosting algorithm. In Gradient Boosting, the model trains decision trees in the background and every tree tries to reduce the error of the previous tree to minimum. Since there is a sequential learning phase, Gradient Boosting model takes more time to work on the dataset. In order to prevent overfitting, I have determined the minimum samples required to be at a leaf node as 10. After training the model accuracy result obtained was 46.2%.

2.6. Feed-Forward Multi Layer Network

In this section of the project, a simple Feed-Forward neural network was implemented using TensorFlow and Keras. After training the data with bagging and boosting methods, the goal was to explore the potential of neural networks for the classification task.

Model Architecture:

- **Input Layer:** The network started with a dense layer of 128 neurons using the ReLU activation function accepting input features from the preprocessed dataset. In order to make the data trainable for the neural network, Standard Scaler is used to normalize the data.
- **Batch Normalizsation and Dropout:** To enhance the model's generalization capability, batch normalization and dropout layers were incorporated after the first dense layer.
- **Hidden Layers:** Two additional hidden layers with 64 and 32 neurons, each followed by batch normalization, were introduced, each using ReLU activation function.
- **Output Layer:** The output layer consisted of neurons equal to the number of unique classes in the target variable, employing the softmax activation function.

Training:

- **Model Compilation:** The model was compiled with the Adam optimizer, sparse categorical entropy loss, and accuracy as the evaluation metric.
- **Training and Validation:** The model underwent training for 10 epochs with a batch size of 32.

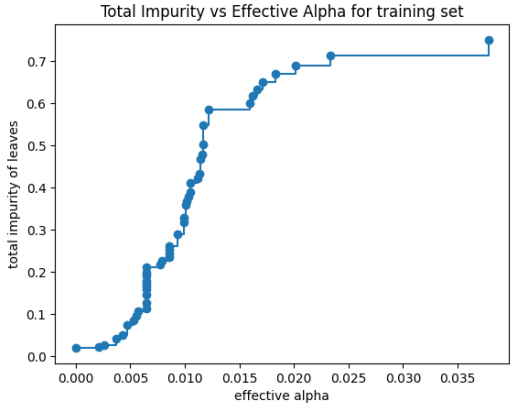
The feed-forward neural network demonstrated its capability to learn and generalize patterns from the data, achieving an accuracy of 48.7% on the test set.

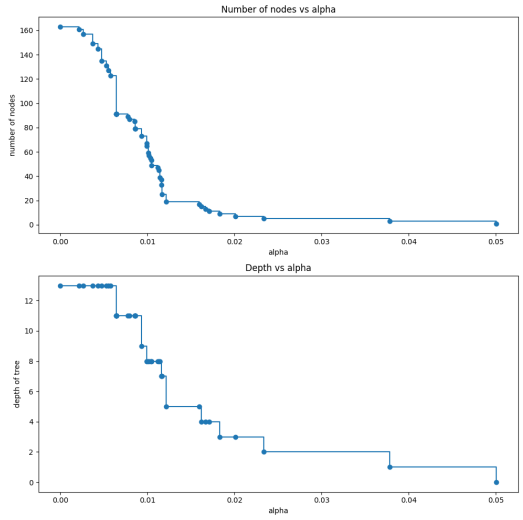
3. Other techniques used for better performance

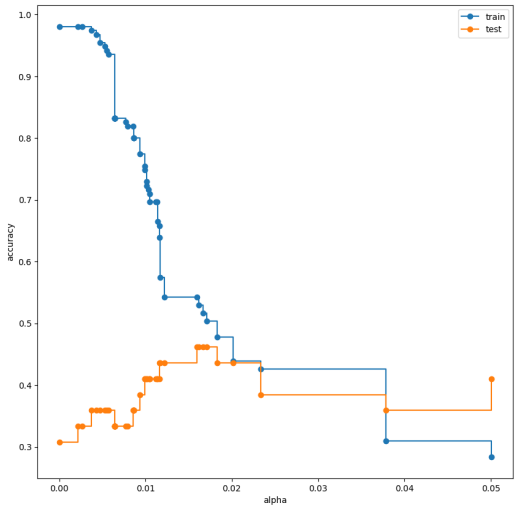
Besides the methods mentioned above, I have used various techniques to obtain better generalization on rules to achieve simpler and more efficient learning models.

3.1. Tree Pruning

Apart from ”min_samples_split” and ”max_depth” parameters that were used to prevent the tree from overfitting, I used cost complexity pruning to control the size of a tree. In the Decision Tree Classifier, this pruning technique is parameterized by the cost complexity parameter, ”ccp_alpha”. If ”ccp_alpha” parameter gets greater, the number of nodes pruned increases, therefore total leaf impurities increase.







After getting ”ccp_alphas” values from the cost complexity pruning path function of the Decision Tree, I made a cross-validated grid search to derive the optimal ccp_alpha value for the tree. Grid search returned the optimal value for ccp_alpha as 0.007 and the accuracy of the Decision Tree Classifier with this parameter was 43.8%.

4. Conclusion

The accuracy of the best performing classifier of each approach can be seen in table below.

Accuracy of classifiers	
Classifier	Score
Decision Tree	43.3%
Random Forest	56.4%
Support Vector Machine	61.5%
Gradient Boosting	46.2%
Feed-Forward NN	43.5%

In this project, the objective was to explore and implement various machine-learning techniques for the classification task using the flag dataset. The dataset, consisting of flag-related features, posed interesting challenges due to its limited size.

4.1. Overview of Approaches

1) **Decision Trees and Random Forests**
The initial analysis involved decision tree-based models. While decision trees provided an intuitive understanding of feature importance, random forests were used to enhance predictive accuracy and potential overfitting with the help of permutation feature importance. The initial decision tree resulted in 33.3% accuracy, with the help of a random forest classifier and the feature

importance, the score has increased to 56.4%. The exploration of decision trees provided insights into feature importance.

2) **Support Vector Machines (SVM)**
Extensive parameter tuning and grid search were conducted to identify optimal hyperparameters, emphasizing the importance of fine-tuning for performance improvement. Various values for gamma and cost parameters were used and the difference between accuracy scores could not be underestimated. With the optimal parameters, the highest accuracy score is reached among other classifiers.

3) **Deep Learning**
Feed-Forward Neural Network was compiled using Adam optimizer with a learning rate of 0.001 and employed sparse categorical cross-entropy as the loss function. The limited dataset size presented challenges in achieving higher accuracy.

This project provided a practical understanding of model selection, hyperparameter tuning, and the complexities associated with working with relatively small datasets. Even though the accuracies achieved with different classifiers are not satisfactory, the increase in the score with hyperparameter-tuning showed how important the model configuration is with respect to the dataset.

During the project, I have learned new techniques to achieve better scores and classifiers that can train the data better. Even though initial models supplied by sklearn and TensorFlow have a simple API, the resources may not be enough to create an efficient model. With this project, I tried to manage to build on the models already provided and obtain better classifiers.