Capstone Project

Project Overview

In light of the recent confederate flag controversy, I realized that flags are not just designs on a piece of cloth. They represent a lot of things to a lot of people and the design of national flags(especially) isn't by accident.

The art and practice of designing flags is known as *vexillography*. All national flags are rectangular, except for the flag of Nepal. The flags of Switzerland and the Vatican City are the only national flags which are exact squares. As of 2011, all national flags consist of at least two different colours. In many cases, the different colours are presented in either horizontal or vertical bands. It is particularly common for colours to be presented in bands of three.

It is common for many flags to feature national symbols, such as coats of arms. National patterns are present in some flags. Variations in design within a national flag can be common in the flag's upper left quarter, or canton.

Problem Statement

The goal here for me was to design a model that could predict the **major religion** (the religion which has the highest percentage of people in that country, affiliated to) of a country based on the the design(by considering factors like number of stripes, types of colors, presence of crosses/crescents/alphabets on the map) of the flag of that country. I also considered other factors like the population, landmass area and geographic zone in my analysis.

I found a dataset on the UCI Machine Learning Repository which had a dataset with information about the flag designs and geographical information for all 194 countries. This is a Supervised Learning - Classification problem as I am looking to predict (classify) the major religion for a country using a dataset that has this data already.

Metrics

My dataset had 194 rows corresponding to each country on the map and 29 feature attributes.
I used 2 evaluation metrics, **Accuracy** and the **F1 score**.

To understand the F1 score, we have to first take a look at precision and recall:

Precision is the fraction of retrieved instances that are relevant, in other words it is the ratio of
`[True Positives/(True Positives + False Positives)]`

Recall(sensitivity) is the fraction of relevant instances that are retrieved, in other words it is the ratio of
`[True Positives/(True Positives + False Negatives)]`

So precision is the ratio of true positives to all positives, while recall is the ratio of true positives to all that were classified correctly. The F1 score is defined as the harmonic mean of precision and recall. I want to ensure that my model is able to predict the religions correctly for each country(precision) while being able to get most of them correct(recall). Hence the F-1 score is ideal for this purpose.

Accuracy simply measures how often the classifier makes the correct prediction. It's the ratio between the number of correct predictions and the total number of predictions (the number of test data points).

Analysis

Data Exploration

I will be using data from the Flags repository in the UCI Machine Learning Repository which is an academic resource for doing Machine Learning experiments. The flags.csv file contains the complete data on every flag from each country along with other topographical data. The variables in the data set are as follows:

Dependent variable(variable we are looking to make our predictions on):

- **religion:** which can take the following values: 0=Catholic, 1=Other Christian, 2=Muslim, 3=Buddhist, 4=Hindu, 5=Ethnic, 6=Marxist, 7=Others

Independent Variables:

- **name**: Name of the country concerned
- **landmass**: 1=N.America, 2=S.America, 3=Europe, 4=Africa, 5=Asia, 6=Oceania
- **zone**: Geographic quadrant, based on Greenwich and the Equator; 1=NE, 2=SE, 3=SW, 4=NW
- **area**: in thousands of square km
- **population**: in round millions

- **language**: 1=English, 2=Spanish, 3=French, 4=German, 5=Slavic, 6=Other Indo-European, 7=Chinese, 8=Arabic, 9=Japanese/Turkish/Finnish/Magyar, 10=Others
- **bars**: Number of vertical bars in the flag
- **stripes**: Number of horizontal stripes in the flag
- **colours**: Number of different colours in the flag
- **red**: 0 if red absent, 1 if red present in the flag
- **green**: same for green
- **blue**: same for blue
- **gold**: same for gold (also yellow)
- **white**: same for white
- **black**: same for black
- **orange**: same for orange (also brown)
- **mainhue**: predominant colour in the flag (tie-breaks decided by taking the topmost hue, if that fails then the most central hue, and if that fails the leftmost hue)
- **circles**: Number of circles in the flag
- **crosses**: Number of (upright) crosses
- **saltires**: Number of diagonal crosses
- **quarters**: Number of quartered sections
- **sunstars**: Number of sun or star symbols
- **crescent**: 1 if a crescent moon symbol present, else 0
- **triangle**: 1 if any triangles present, 0 otherwise
- **icon**: 1 if an inanimate image present (e.g., a boat), otherwise 0
- **animate**: 1 if an animate image (e.g., an eagle, a tree, a human hand) present, 0 otherwise
- **text**: 1 if any letters or writing on the flag (e.g., a motto or slogan), 0 otherwise
- **topleft**: colour in the top-left corner (moving right to decide tie-breaks)
- **botright**: Colour in the bottom-left corner (moving left to decide tie-breaks)

Some of the general statistics for this dataset are as follows:

```
       landmass        zone          area   population    language  \
count  194.000000  194.000000    194.000000   194.000000  194.000000
mean     3.572165    2.211340    700.046392    23.268041    5.340206
std      1.553018    1.308274   2170.927932    91.934085    3.496517
min      1.000000    1.000000      0.000000     0.000000    1.000000
25%      3.000000    1.000000      9.000000     0.000000    2.000000
50%      4.000000    2.000000    111.000000     4.000000    6.000000
75%      5.000000    4.000000    471.250000    14.000000    9.000000
max      6.000000    4.000000  22402.000000  1008.000000   10.000000

       religion        bars     stripes     colours         red     ...     \
count  194.000000  194.000000  194.000000  194.000000  194.000000     ...
mean     2.190722    0.453608    1.551546    3.463918    0.788660     ...
```

```
std      2.061167    1.038339    2.328005    1.300154    0.409315     ...
min      0.000000    0.000000    0.000000    1.000000    0.000000     ...
25%      1.000000    0.000000    0.000000    3.000000    1.000000     ...
50%      1.000000    0.000000    0.000000    3.000000    1.000000     ...
75%      4.000000    0.000000    3.000000    4.000000    1.000000     ...
max      7.000000    5.000000   14.000000    8.000000    1.000000     ...

         saltires    quarters    sunstars    crescent    triangle       icon  \
count  194.000000  194.000000  194.000000  194.000000  194.000000  194.000000
mean     0.092784    0.149485    1.386598    0.056701    0.139175    0.252577
std      0.290879    0.435860    4.396186    0.231869    0.347025    0.435615
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
50%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
75%      0.000000    0.000000    1.000000    0.000000    0.000000    0.750000
max      1.000000    4.000000   50.000000    1.000000    1.000000    1.000000

         animate        text     topleft    botright
count  194.000000  194.000000  194.000000  194.000000
mean     0.201031    0.082474    1.731959    1.432990
std      0.401808    0.275798    1.603206    1.529721
min      0.000000    0.000000    0.000000    0.000000
25%      0.000000    0.000000    0.000000    0.000000
50%      0.000000    0.000000    1.000000    1.000000
75%      0.000000    0.000000    3.000000    2.000000
max      1.000000    1.000000    7.000000    7.000000
```

Here is a sample of what the actual dataset looks like(there are 194 rows and 30 columns but it I have shortened it for ease of readability):

	name	landmass	zone	area	population	language	religion	bars	stripes	colours	...
0	Afghanistan	5	1	648	16	10	2	0	3	5	...
1	Albania	3	1	29	3	6	6	0	0	3	...
2	Algeria	4	1	2388	20	8	2	2	0	3	...
3	American-Samoa	6	3	0	0	1	1	0	0	5	...
4	Andorra	3	1	0	0	6	0	3	0	3	...

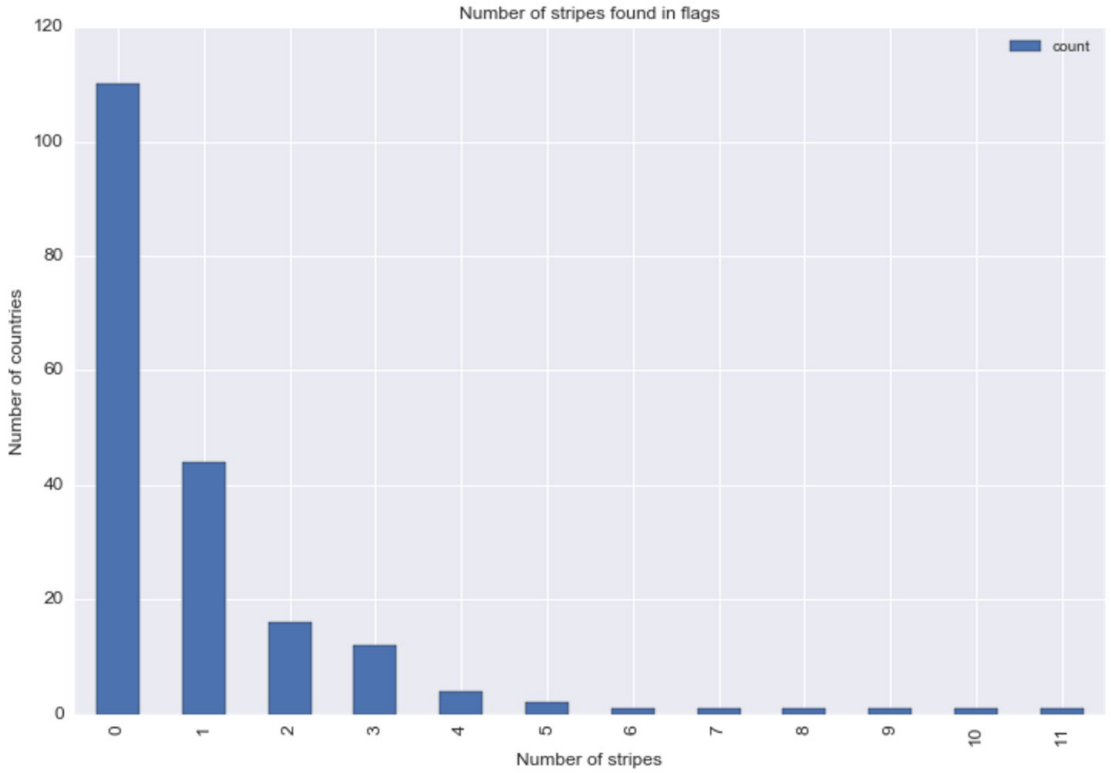
Looking at the statistical summary of the data I can infer the following:

- Most countries tend to have at least 1 stripe on their flag.
- Flags in general tend to have some combination of 3 colors in them.
- Christianity or some variant of it is the most popular religion and is spread as the major religion in most countries.
- Having bars in the flag is not very common, but this statistic comes with a slightly higher standard deviation which leads me to think that some flags don't have bars at all while others tend to have 2 or more.
- Saltiers(diagonal crosses) are extremely rare in flags.
- Having the symbol of a sun or a star on the flag comes with an extremely high standard deviation which leads me to think while some flags do not have them at all, others tend to have a high number of them in their flag, hence this statistic is spread out.
- Crescents and crosses which tend to be associated with religion, are not very commonly found in flags.
- Animate image (e.g., an eagle, a tree, a human hand) are also not commonly found in most flags.
- Having text written on the flag is not common either.

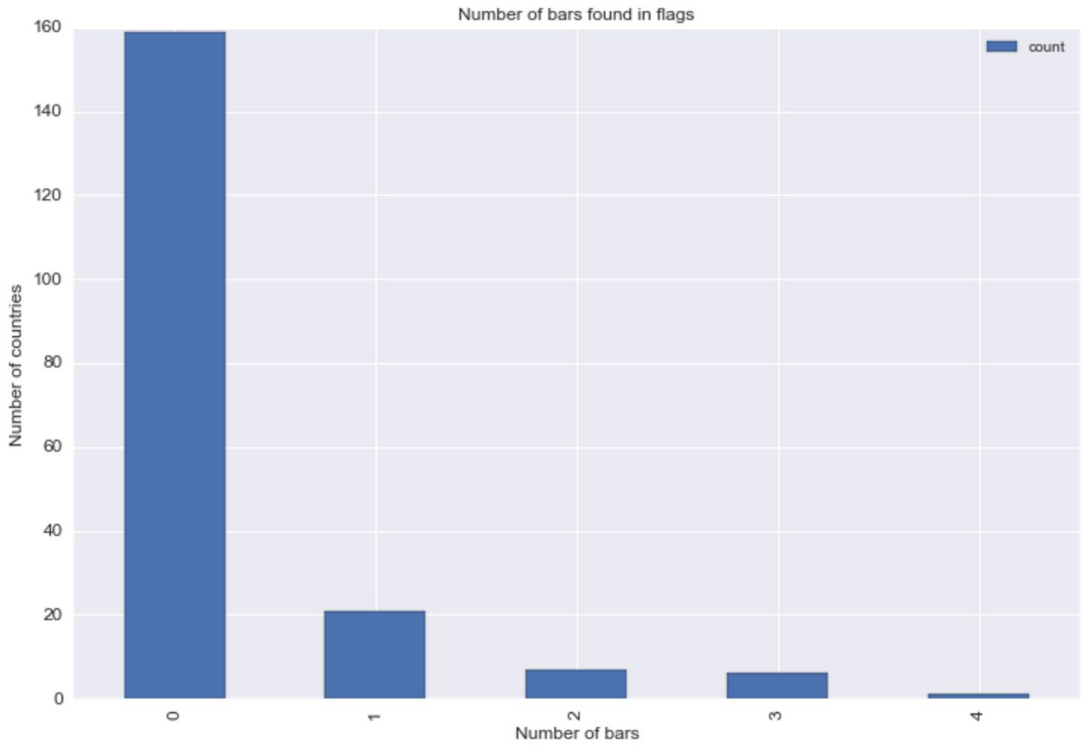
Exploratory Visualization

The two most striking features in most flags today are horizontal stripes and vertical bars and therefore that is where I decided to start.

I started by checking what was the trend in terms of the number of stripes(horizontal stripes) on the flag for all the countries. Interestingly, most countries have <4 stripes in their flag.



I then checked what was the trend in terms of the number of bars(vertical bars) on the flag for all the countries. This had a lower range than compared to the horizontal stripes. Also, interestingly a lot of countries do not have vertical bars at all.



Algorithms and Techniques

This is a classification problem and therefore I used Support Vector Machines, Gaussian Naive Bayes, Decision Trees, Random Forest Trees and Adaboost Trees to run my model.

Support Vector Machines:

SVMs models the features as points in space and tries to form boundary lines between different classes of variables, thus separating each class of features into different planes. Its strength is in dividing features that have a clear separation. It can quickly separate different classes of features when they are distinctively apart.

Advantages of using SVMs are:

- Training is relatively easy and it has no local optimal, unlike in neural networks.
- Effective in high dimensional spaces. Effective in cases where number of dimensions is greater than the number of samples.
- It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

- In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.
- Non-traditional data like strings and trees can be used as input to SVM, instead of feature vectors

Disadvantage of using SVMs are:

- It tends to fails in situations when points are uniformly distributed.
- One issue with SVMs is having to choose a kernel function.
- If the number of features is much greater than the number of samples, the method is likely to give poor performances.
- Prone to overfitting.

SVM is suitable for current scenario, because we need to perform a classification of countries to religions and also because the number of features are also not that large as compared our sample size.

Gaussian Naive Bayes:

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations. I used GNB() for my problem as the features I am using can be seen to be independant of each other, albeit that is a naive assumption to make. T

Advantages of GNB():

- They require a small amount of training data to estimate the necessary parameters.
- Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods.
- The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution.
- This in turn helps to alleviate problems stemming from the curse of dimensionality.

Disadvantages of using naive Bayes:

- Although naive Bayes is known as a decent classifier, it is known to be a bad estimator.

Decision Trees:

Decision Trees (DTs) are a non-parametric supervised learning method. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. As I have a supervised classification problem, I used them to predict the religion classification based on a set of features.

Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable. See algorithms for more information.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.

- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Random Forest Trees:

Random Forest is an ensemble model of decision trees. It is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Strengths:

- Very good at handling binary features.
- Good at handling large volumes of data.

Weaknesses:

- Tendency to overfit.
- Lower prediction accuracy.
- Tendency to have a high variance.

I used this algorithm for this dataset as Random Forest classifiers tend to perform well with supervised classification problems.

Benchmark

For this dataset, there is no explicit benchmark specified as is usually the case with most ML datasets as they are highly specific in nature. I was looking to get an F1 score of greater than 0.5 as in most situations, you have a trade-off between precision and recall. If you optimize your classifier to increase one and disfavor the other, the harmonic mean quickly decreases. It is greatest however, when both precision and recall are equal.

Methodology

Data Preprocessing

I had to do a little bit of pre-processing on the data set to convert certain text based features(categorical variable) to numerical values. I did this for the ‘botright’ (colour in the bottom-left corner), ‘topleft’ (colour in the top-left corner and the ‘mainhue’ (predominant colour in the flag) columns. I translated their values from text to numeric by creating dummy variables for each of them as follows:

```
df_topleft = pd.get_dummies(data['topleft']).rename(columns=lambda x:
'topleft_' + str(x))
```

I did it in a similar way for the 'botright' and 'mainhue' columns. There are no missing values in this data set.

Outliers: I did some analysis on outlier detection using Tukey's method:

With Tukey's method, outliers are:

values below (Quartile 1) – (1.5 × IQR)
values above (Quartile 3) + (1.5 × IQR)

I found that although on paper there were a large number of outliers, this is actually not the case. Flags are unique in nature and have pretty distinctive features. Considering the features of a flag that classify it to be an outlier are the same features that give it its distinctive look. This is a classic example of 'domain knowledge' where having some level of background knowledge can be key in ensuring you are treating your data properly. Therefore, keeping this in mind, I did not remove any of the flags from our dataset albeit a lot of them have 2 to 3 features which classify it to be as an outlier.

I used the `MinMaxScaler` to scale two features, `area` and `population`. These features have a very wide and different range, standard deviation as well as percentile ranges. The data is scaled to a fixed range - 0 to 1. I am doing this so that we end up with smaller standard deviations, which can suppress the effect of outliers.

I applied the scaler on the two columns as follows:

```
data[['area','population']] =
data[['area','population']].apply(lambda x:
MinMaxScaler().fit_transform(x))
```

Implementation

Before I ran my classifier, I performed **feature selection** to check which features are the most likely to affect my prediction.

In order to do this I used to two helper functions, `featureFormat` which takes a list of features ('features_list'), searches the data dictionary for those features, and returns those features in the form of a numpy array data list. The second helper function I used is `targetFeatureSplit`, which splits the data list, returned by `featureFormat`, into religion(target) and features as their own separate lists. I fed this into

`SelectKBest()` which gave me the best and most influential features to predict religion. The features along with their scores is as follows:

```
[('language', 13.388135105449459),
 ('landmass', 10.39438223356032),
 ('zone', 8.9322966956954453),
 ('crosses', 8.8412934900390834),
 ('blue', 6.4287074621657823),
 ('green', 6.0348972262058336),
 ('saltires', 5.4011022594922888),
 ('crescent', 4.8289550621319579),
 ('botright_green', 4.6300591502082042),
 ('topleft_white', 4.416168473817117),
 ('botright_red', 3.9503165975516445),
 ('population', 3.8741220801625889),
 ('botright_black', 3.8726368122975874),
 ('mainhue_blue', 3.8610157349865131),
 ('mainhue_white', 3.7882942213834432),
 ('topleft_green', 3.4911005424270716),
 ('botright_blue', 3.2680717671973607),
 ('mainhue_green', 3.1260971561259137),
 ('circles', 2.6831434124637852),
 ('animate', 2.5821008525379097),
 ('mainhue_orange', 2.500667823694545),
 ('topleft_orange', 2.500667823694545),
 ('colours', 2.4570123986525831),
 ('quarters', 2.2151781868810505),
 ('gold', 2.2147478132285521),
 ('white', 2.184391258254887),
 ('orange', 1.9263213750312018),
 ('mainhue_red', 1.7472946857298732),
 ('topleft_gold', 1.7335102137179734),
 ('botright_white', 1.6800192939661023),
 ('topleft_blue', 1.5897965858325955),
 ('bars', 1.5890468348678537),
 ('area', 1.5148208706410931),
 ('topleft_red', 1.5110183037637455),
 ('black', 1.3694930895992941),
 ('stripes', 1.2994746362255749),
 ('icon', 1.1673841786746191),
 ('red', 1.1302639240152668),
 ('mainhue_gold', 0.89215738115070553),
```

```
 ('sunstars', 0.73759874552528337),
 ('triangle', 0.70930627676102098),
 ('botright_gold', 0.68839311693275951),
 ('topleft_black', 0.65006976595272581),
 ('botright_orange', 0.54084060269627288),
 ('text', 0.49828106143143597),
 ('mainhue_black', 0.33668093116611225)]
```

The top features obtained are:

```
['blue', 'mainhue_white', 'crosses', 'language', 'zone', 'topleft_white',
'botright_red', 'mainhue_blue', 'landmass', 'green', 'crescent',
'saltires', 'population', 'botright_black', 'botright_green']
```

I ran my classifiers using the top 15 features that I got using `SelectKBest()` and calculated the mean values of Accuracy, Precision and Recall over these iterations. I validated my data by running through my algorithm 500 times to get mean values for accuracy, precision and recall for each algorithm. The classic mistake of validation is over fitting where our algorithm performs well when we consider our training data but not well with the test data.

Refinement

The result I got as part of **Gaussian Naive Bayes** analysis is as follows:

```
Accuracy:  0.2542372
Precision:  0.498486
Recall:  0.25423728
```

Seeing these low values, I did not proceed with tuning and went ahead with my analysis.

The result I got as part of **Support Vector Machines** analysis is as follows:

```
Accuracy:  0.474576271186
Precision:  0.43731503901
Recall:  0.474576271186
```

Considering the low values, I did not proceed to fine tune the algorithm and went ahead with my analysis.

The result I got as part of **Decision Tree Classifier** analysis is as follows:

```
Accuracy:  0.534203389831
Precision:  0.56123818999
Recall:  0.534203389831
```

I fine tuned my algorithm by setting up a parameter grid to get the possible F1 score. I used `StratifiedShuffleSplit` to split my data. Considering the scores I got for the Decision Tree classifier and Random Forest classifier were above my 0.5 benchmark threshold, I went ahead and fine tuned them.

I tuned my Decision Tree classifier as follows:

```
dtc = GridSearchCV(DecisionTreeClassifier(),param_grid = {'criterion':
['gini', 'entropy'],'splitter': ['best' ,'random'],'max_features':
['sqrt', 'auto', 'log2'],'presort' : ['True', 'False'],'random_state' :
[None, 1, 2, 3, 4, 5, 6, 7, 8, 9]},cv = cv,scoring = 'f1')
```

The best estimator obtained was:

```
DecisionTreeClassifier best estimator:
DecisionTreeClassifier(class_weight=None, criterion='gini',
max_depth=None,
            max_features='sqrt', max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort='True', random_state=4, splitter='best')
```

The **best estimator** is the estimator that was chosen by the search, i.e. estimator which gave highest score (or smallest loss if specified) on the left out data.
The **best score** is the score of best_estimator on the left out data.
The **best params** is the parameter setting that gave the best results amongst all the pararmeters specified to try out.

The best parameter values obtained were:

```
DecisionTreeClassifier best parameters:  {'max_features': 'sqrt',
'presort': 'True', 'random_state': 4, 'criterion': 'gini', 'splitter':
'best'}
DecisionTreeClassifier best score:  0.659267010767
```

Upon fine tuning my algorithm, I was able to get a final F-1 score of **0.659267010767**

I performed a similar analysis with **AdaBoost** classifier and upon fine tuning got an F-1 score of **0.57.**

The result I got as part of **Random Forest Tree Classifier** analysis is as follows:

```
Accuracy:  0.601661016949
Precision:  0.595313166938
Recall:  0.601661016949
```

I tuned by Random forest classifier as follows:

```
rfc = GridSearchCV(RandomForestClassifier(),param_grid = {'n_estimators':
[200, 700],'max_features': ['auto', 'sqrt', 'log2']}, cv = cv,scoring =
'f1')
```

The best estimator obtained was:

```
RandomForestClassifier best estimator:
RandomForestClassifier(bootstrap=True, class_weight=None,criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

The best parameter values obtained were:

```
RandomForestClassifier best parameters:  {'max_features': 'auto'
'n_estimators': 200}
```

Upon fine tuning my algorithm, the final F-1 score I got was: **0.725227772228** which I was quite happy with!

The fine tuned parameters of my algorithm and what they mean is as follows:

Parameter	Explanation
Bootstrap = True	The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True.
Class_weight = None	Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one.
Criterion = gini	The function to measure the quality of a split.
Max_depth = None	The maximum depth of the tree

Max_features = auto	The number of features to consider when looking for the best split
Max_leaf_nodes = None	Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity.
Min_samples_leaf = 1	The minimum number of samples in newly created leaves. A split is discarded if after the split, one of the leaves would contain less than min_samples_leaf samples.
Min_samples_split = 2	The minimum number of samples required to split an internal node
Min_weight_fraction_leaf = 0.0	The minimum weighted fraction of the input samples required to be at a leaf node
N_estimators = 200	The number of trees in the forest.
N_jobs = 1	The number of jobs to run in parallel for both fit and predict.
Oob_score = False	Whether to use out-of-bag samples to estimate the generalization error.
Random_state = None	The random number generator is the RandomState instance used by np.random.
Verbose = 0	Controls the verbosity of the tree building process
Warm_start = False	When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest.

Results

Model Evaluation and Validation

The final F-1 scores for for my models after tuning them were as follows:

Algorithm	F1 Score
Decision Tree Classifier	0.659
AdaBoost Classifier	0.57
Random Forest Classifier	**0.725**

The Random Forest Classifier algorithm came out on top with the score 0.725 and that is what I decided to base my model on. The training and prediction times were similar to all the algorithms I tested and is hence irrelevant to my evaluation.

Validation: I ran my algorithm using the Stratified Shuffle Split mechanism for splitting my data. Stratified random sampling involves first dividing a population into subpopulations and then applying random sampling methods to each subpopulation to form a test group. The biggest advantage of stratified random sampling is that it reduces selection bias which could occur if I explicitly took out data to test for in the end. Stratifying the entire population before applying random sampling methods helps ensure a sample that accurately reflects the population being studied in terms of the criteria used for stratification. Stratified random sampling is also advantageous when it can be used accurately because it ensures each subgroup within the population receives proper representation within the sample. Because of this, I can say that my results will work well with unseen data as well.

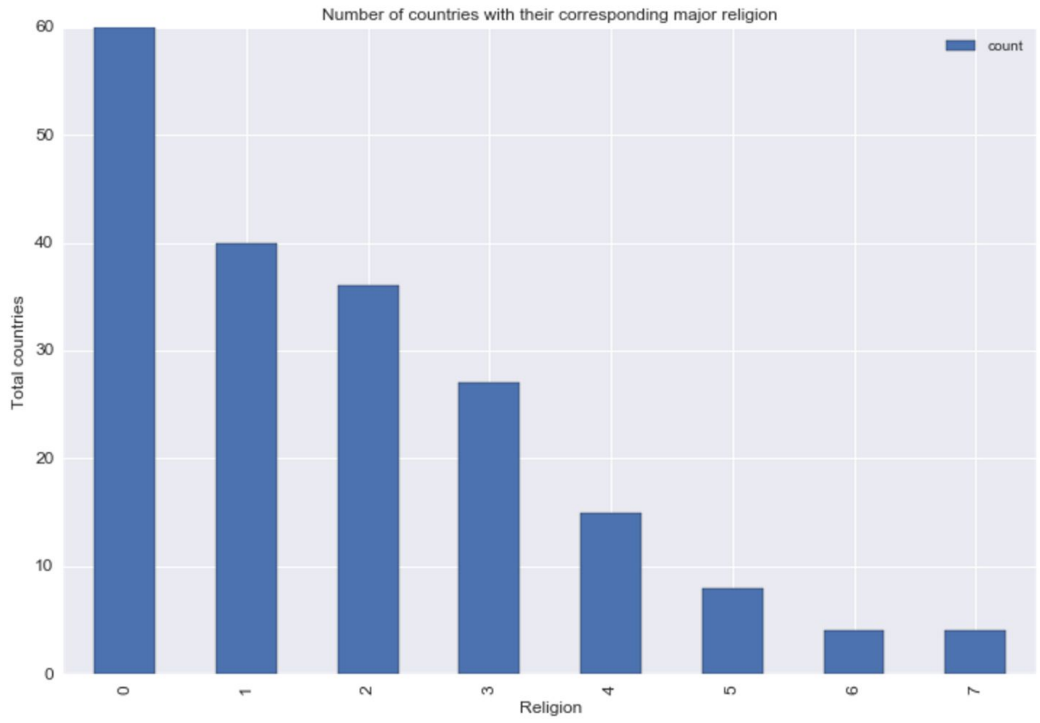
Justification

The model I used has an F1 score of **0.725** which is higher than the 0.5 threshold value I had set in the benchmarking phase.

Conclusion

Free Form Visualization

I analysed how many countries have as their major religion Christianity, Islam etc. I used value_counts() to find the total number of countries and indexed them by their major religion.



To put this in perspective graphically, I plotted a bar plot of the number of countries you can find the followers of various religions sorted in descending order: (key for religion: 0=Catholic, 1=Other Christian, 2=Muslim, 3=Buddhist, 4=Hindu, 5=Ethnic, 6=Marxist, 7=Others)

As can be seen in the bar plot above, if there is one thing to take away from this project it is that Christianity is by far the most widespread religion, both in terms of number of followers and in term of its spread across different countries. Islam comes in second, followed by the Ethnic and Marxist religions. Hinduism and Buddhism although have an extremely high number of followers(most of which are in India, China and other East Asian countries) do not have as much spread in terms the number of countries you can find their followers in.

Reflection

In order to predict the the majority religion of a nation based on its flag design and other topographical factors, I started with a thorough analysis of my data to check for any pre processing requirements.
After taking care of that, my next step was feature selection where I used an algorithm to select the most relevant and best features to predict the majority religion.
Once I had my best features, I dove into a few supervised classification algorithms and tested them by fine tuning a wide variety of parameters by creating a parameter grid. I picked out the best parameters from my parameter grid and got my final model which gave my an F1 score of 0.725. The F1 score is the harmonic mean of the Precision and Recall

scores. Precision, with regards to this experiment, is the ability of my algorithm to correctly classify the religion of a country based on the flag features being fed into the algorithm. Recall, on the other hand is the ability of the algorithm to correctly classify *most* of the country's' major religion correctly. If I optimize my classifier to increase one and disfavor the other, the harmonic mean quickly decreases. It is greatest however, when both precision and recall are equal. So an F1 score of 0.725 means that the algorithm can not only perform the classification correctly for the most part, but it does so for most of countries.

Improvement

Using information about the flag and topography of a country to predict it's majority religion could be improved if I had more data about the demographics of the countries. Race alignment with religion could be an interesting metric to look at.
Also, having more historical data about religions of the past that are not very popular anymore could be interesting to look at if I facet wrapped that data with a time metric to show the rise and fall of popularity of certain religions, for example, religions like Zoroastrianism which are not as popular anymore but used to be in the past.
Another improvement can be made if I had access to the changing design of flags. Certain flags were redesigned when they those nations were colonised and having a look at the original designs could definitely help my analysis as well.