Introduction

This dataset contains details of various nations and their flags. With this data we are trying to predict the religion of a country from the shapes and the colours in its flag.

Part II- Modelling Phase

In [179]:

```
path="C:\\Users\\user\\Assignment 2"
```

In [180]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [181]:

```
flags = pd.read_csv(path+"/part1.csv", delimiter=';')
```

Generating Train and Test set

In [183]:

```
from sklearn.model_selection import train_test_split
```

In [184]:

```
print(flags.groupby('religion').size()) # to check the biasness of the target variable
```

```
religion
0    40
1    60
2    36
3     8
4     4
5    27
6    15
7     4
dtype: int64
```

In [185]:

```
flags1 = flags.copy()
flags1.pop('Unnamed: 0')
flags1.pop('religion1')
flags1.head(2)
```

Out[185]:

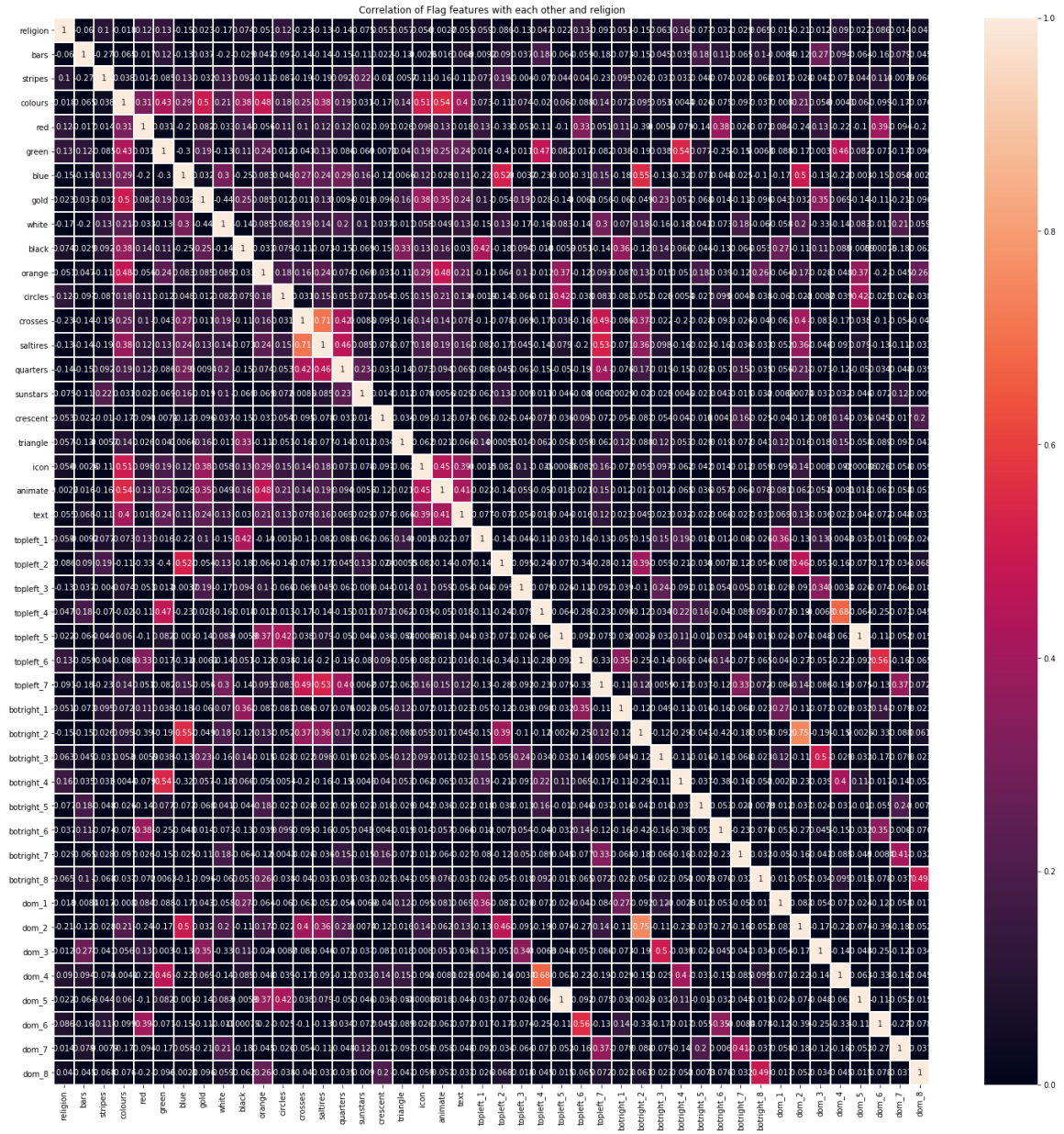
	religion	bars	stripes	colours	red	green	blue	gold	white	black	...	botright_7	botr
0	2	0	3	5	1	1	0	1	1	1	...	0	
1	6	0	0	3	1	0	0	1	0	1	...	0	

2 rows × 44 columns

Correlation among different features and between different features and religion

In [186]:

```
import seaborn as sns
data_corr = flags1.corr()
fig, ax = plt.subplots(figsize=(25,25))
sns.heatmap(data_corr , vmin=0, vmax=1, square=False, annot=True, linewidths=1, ax=ax)
plt.title("Correlation of Flag features with each other and religion")
plt.show()
```


Correlation of Flag features with each other and religion
religion
bars
stripes
colours
red
green
blue
gold
white
black
orange
circles
crosses
saltires
quarters
sunstars
crescent
triangle
icon
animate
text
topleft_1
topleft_2
topleft_3
topleft_4
topleft_5
topleft_6
topleft_7
botright_1
botright_2
botright_3
botright_4
botright_5
botright_6
botright_7
botright_8
dom_1
dom_2
dom_3
dom_4
dom_5
dom_6
dom_7
dom_8
1.0
0.8
0.6
0.4
0.2
0.0

In [187]:

```
flags1.head(2)
```

Out[187]:

	religion	bars	stripes	colours	red	green	blue	gold	white	black	...	botright_7	botr
0	2	0	3	5	1	1	0	1	1	1	...	0	
1	6	0	0	3	1	0	0	1	0	1	...	0	

2 rows × 44 columns

In [71]:

```
flags1.columns
```

Out[71]:

```
Index([u'religion', u'bars', u'stripes', u'colours', u'red', u'green', u'b
lue',
       u'gold', u'white', u'black', u'orange', u'circles', u'crosses',
       u'saltires', u'quarters', u'sunstars', u'crescent', u'triangle',
       u'icon', u'animate', u'text', u'topleft_1', u'topleft_2', u'topleft
_3',
       u'topleft_4', u'topleft_5', u'topleft_6', u'topleft_7', u'botright_
1',
       u'botright_2', u'botright_3', u'botright_4', u'botright_5',
       u'botright_6', u'botright_7', u'botright_8', u'dom_1', u'dom_2',
       u'dom_3', u'dom_4', u'dom_5', u'dom_6', u'dom_7', u'dom_8'],
      dtype='object')
```

In [72]:

```
#X=flags1.iloc[:,[0,1,2,3,4,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25]]
#y= flags1.iloc[:,5]
y=flags1.pop('religion')
X=flags1
```

In [73]:

```
#X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.80,random_state=
4)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=
42, stratify=y) # stratified sampling
```

In [74]:

```
X_train.shape
```

Out[74]:

```
(155, 43)
```

In [75]:

```
y_train.shape
```

Out[75]:

```
(155L,)
```

In [76]:

```
X_test.shape
```

Out[76]:

```
(39, 43)
```

In [77]:

```
y_test.shape
```

Out[77]:

```
(39L,)
```

Data Modelling

KNN

In [78]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [79]:

```
clf1 = KNeighborsClassifier(5) # Default value
```

In [80]:

```
clf1.fit(X_train, y_train)
```

Out[80]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

In [81]:

```
predicted1 = clf1.predict(X_test)
```

In [82]:

```
predicted1
```

Out[82]:

```
array([1, 1, 2, 1, 2, 1, 0, 0, 0, 2, 2, 0, 0, 1, 1, 1, 1, 2, 5, 1, 1, 5,
       1, 1, 0, 0, 1, 2, 1, 1, 5, 5, 0, 0, 1, 0, 1, 0, 5], dtype=int64)
```

In [83]:

```
clf1.score(X_test, y_test)
```

Out[83]:

```
0.5897435897435898
```

Using model complexity curve to determine the best K-value for testing accuracy

In [84]:

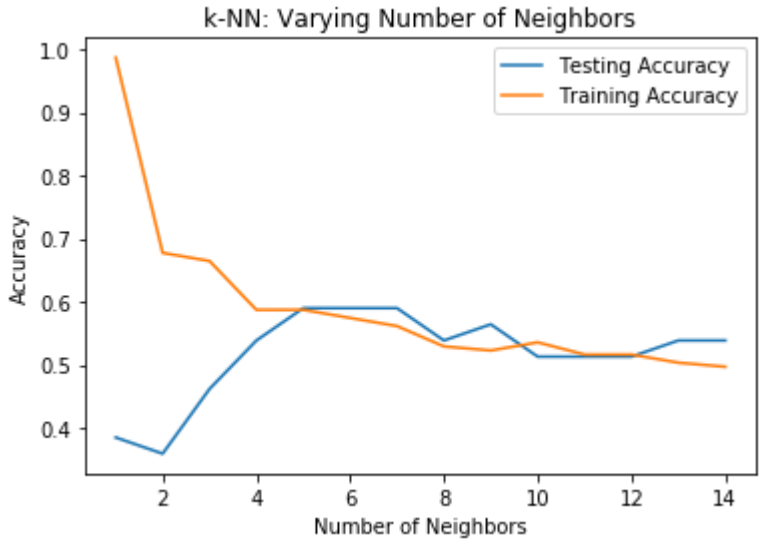
```
neighbors = np.arange(1, 15)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
for i, k in enumerate(neighbors):
    # Setup a k-NN Classifier with k neighbors: knn
    knn = KNeighborsClassifier(n_neighbors=k)

    # Fit the classifier to the training data
    knn.fit(X_train, y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train)

    #Compute accuracy on the testing set
    test_accuracy[i] = knn.score(X_test, y_test)

# Generate plot
plt.title('k-NN: Varying Number of Neighbors')
plt.plot(neighbors, test_accuracy, label = 'Testing Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
```



In [85]:

```
# Test accuracy is highest for K=7
```

In [86]:

```
clf2 = KNeighborsClassifier(7)
```

In [87]:

```
clf2.fit(X_train, y_train)
```

Out[87]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=7, p=2,
           weights='uniform')
```

In [88]:

```
predicted2 = clf2.predict(X_test)
```

In [89]:

```
clf2.score(X_test, y_test)
# print "[Train/test split] score: {:.5f}".format(clf.score(X_test, y_test))
```

Out[89]:

```
0.5897435897435898
```

k-NN classifier with 11 neighbors has learned from the training data and predicted the labels of the religions in the test set with 49% accuracy

In [90]:

```
from sklearn.metrics import confusion_matrix
```

In [91]:

```
predicted2.shape
```

Out[91]:

```
(39L,)
```

In [92]:

```
cm2 = confusion_matrix(y_test,predicted2)
```

In [93]:

```
print cm2
```

```
[[ 6  1  1  0  0  0  0  0]
 [ 0 12  0  0  0  0  0  0]
 [ 1  2  3  0  0  1  0  0]
 [ 1  1  0  0  0  0  0  0]
 [ 1  0  0  0  0  0  0  0]
 [ 1  1  1  0  0  2  0  0]
 [ 2  0  1  0  0  0  0  0]
 [ 0  0  0  0  0  1  0  0]]
```

K-fold cross validation

In [94]:

```
from sklearn.model_selection import KFold
```

In [95]:

```
kf = KFold(n_splits=6 ,random_state=42)
```

In [96]:

```
#kf.get_n_splits(flags1)
```

In [97]:

```
#KFold(n_splits=5, random_state=None, shuffle=False)
```

In [98]:

```
from sklearn.metrics import classification_report
```

In [103]:

```
print classification_report(y_test,predicted2)
```

```
ValueErrorTraceback (most recent call last)
<ipython-input-103-0bed64f45cc7> in <module>()
----> 1 print classification_report(y_test,predicted2)

C:\Users\user\Anaconda2\lib\site-packages\sklearn\metrics\classification.p
yc in classification_report(y_true, y_pred, labels, target_names, sample_w
eight, digits)
   1444                                                labels=labels,
   1445                                                average=None,
-> 1446                                                sample_weight=sa
mple_weight)
   1447
   1448     row_fmt = u'{:>{width}s} ' + u' {:>9.{digits}f}' * 3 + u' {:>
9}\n'

C:\Users\user\Anaconda2\lib\site-packages\sklearn\metrics\classification.p
yc in precision_recall_fscore_support(y_true, y_pred, beta, labels, pos_la
bel, average, warn_for, sample_weight)
   1023         raise ValueError("beta should be >0 in the F-beta score")
   1024
-> 1025     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
   1026     present_labels = unique_labels(y_true, y_pred)
   1027

C:\Users\user\Anaconda2\lib\site-packages\sklearn\metrics\classification.p
yc in _check_targets(y_true, y_pred)
     69     y_pred : array or indicator matrix
     70     """
---> 71     check_consistent_length(y_true, y_pred)
     72     type_true = type_of_target(y_true)
     73     type_pred = type_of_target(y_pred)

C:\Users\user\Anaconda2\lib\site-packages\sklearn\utils\validation.pyc in
check_consistent_length(*arrays)
    202     if len(uniques) > 1:
    203         raise ValueError("Found input variables with inconsistent
 numbers of"
--> 204                          " samples: %r" % [int(l) for l in length
s])
    205
    206

ValueError: Found input variables with inconsistent numbers of samples: [3
3, 39]
```

In [104]:

```
for train_index,test_index in kf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

```
('TRAIN:', array([ 33,  34,  35,  36,  37,  38,  39,  40,  41,  42,  43,
44,  45,
        46,  47,  48,  49,  50,  51,  52,  53,  54,  55,  56,  57,  58,
        59,  60,  61,  62,  63,  64,  65,  66,  67,  68,  69,  70,  71,
        72,  73,  74,  75,  76,  77,  78,  79,  80,  81,  82,  83,  84,
        85,  86,  87,  88,  89,  90,  91,  92,  93,  94,  95,  96,  97,
        98,  99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110,
       111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123,
       124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136,
       137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149,
       150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162,
       163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175,
       176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188,
       189, 190, 191, 192, 193]), 'TEST:', array([ 0,  1,  2,  3,  4,  5,
6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]))
('TRAIN:', array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,
11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  66,  67,  68,  69,  70,  71,
        72,  73,  74,  75,  76,  77,  78,  79,  80,  81,  82,  83,  84,
        85,  86,  87,  88,  89,  90,  91,  92,  93,  94,  95,  96,  97,
        98,  99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110,
       111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123,
       124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136,
       137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149,
       150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162,
       163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175,
       176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188,
       189, 190, 191, 192, 193]), 'TEST:', array([33, 34, 35, 36, 37, 38,
39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
       50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65]))
('TRAIN:', array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,
11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  98,  99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109,
       110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122,
       123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135,
       136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148,
       149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161,
       162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174,
       175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187,
       188, 189, 190, 191, 192, 193]), 'TEST:', array([66, 67, 68, 69, 70,
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
       83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97]))
('TRAIN:', array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,
11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97, 130, 131, 132, 133, 134, 135,
       136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148,
       149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161,
       162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174,
       175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187,
```

```
       188, 189, 190, 191, 192, 193]), 'TEST:', array([ 98,  99, 100, 101,
102, 103, 104, 105, 106, 107, 108, 109, 110,
       111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123,
       124, 125, 126, 127, 128, 129]))
('TRAIN:', array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,
11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
       104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
       117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
       162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174,
       175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187,
       188, 189, 190, 191, 192, 193]), 'TEST:', array([130, 131, 132, 133,
134, 135, 136, 137, 138, 139, 140, 141, 142,
       143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
       156, 157, 158, 159, 160, 161]))
('TRAIN:', array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,
11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
       104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
       117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
       130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
       143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
       156, 157, 158, 159, 160, 161]), 'TEST:', array([162, 163, 164, 165,
166, 167, 168, 169, 170, 171, 172, 173, 174,
       175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187,
       188, 189, 190, 191, 192, 193]))
```

In [105]:

```
for k, (train_index, test_index) in enumerate(kf.split(X)):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    clf.fit(X_train, y_train)
    print "[fold {0}] score: {1:.5f}".format(k, clf.score(X_test, y_test))
```

```
NameErrorTraceback (most recent call last)
<ipython-input-105-3e5438ae3c67> in <module>()
      2     X_train, X_test = X.iloc[train_index], X.iloc[test_index]
      3     y_train, y_test = y.iloc[train_index], y.iloc[test_index]
----> 4     clf.fit(X_train, y_train)
      5     print "[fold {0}] score: {1:.5f}".format(k, clf.score(X_test,
 y_test))

NameError: name 'clf' is not defined
```

Best score value for 5-fold cross validation

Parameter Tuning Effect

While we keep 'n_neighbors = 11', we set "weights='distance'".

In [111]:

```
clf3 = KNeighborsClassifier(7, weights='distance')
```

In [112]:

```
fit = clf3.fit(X_train, y_train)
```

In [113]:

```
predicted3 = fit.predict(X_test)
```

In [114]:

```
clf3.score(X_test, y_test)
```

Out[114]:

```
0.42424242424242425
```

In [115]:

```
cm3 = confusion_matrix(y_test, predicted3)
```

In [116]:

```
print classification_report(y_test,predicted3)
```

```
             precision    recall  f1-score   support

          0       0.29      0.25      0.27         8
          1       0.58      0.64      0.61        11
          2       0.33      0.60      0.43         5
          3       0.00      0.00      0.00         2
          5       0.33      0.20      0.25         5
          6       1.00      0.50      0.67         2
          7       0.00      0.00      0.00         0

avg / total       0.43      0.42      0.41        33


C:\Users\user\Anaconda2\lib\site-packages\sklearn\metrics\classification.p
y:1135: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\user\Anaconda2\lib\site-packages\sklearn\metrics\classification.p
y:1137: UndefinedMetricWarning: Recall and F-score are ill-defined and bei
ng set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
```

Let's further set 'p' value to 1 # Manhattan Distance

In [117]:

```
clf4 = KNeighborsClassifier(9, weights='distance', p=1)
```

In [118]:

```
fit = clf4.fit(X_train, y_train)
```

In [119]:

```
predicted4 = fit.predict(X_test)
```

In [120]:

```
clf4.score(X_test, y_test)
```

Out[120]:

```
0.45454545454545453
```

In [121]:

```
cm4 = confusion_matrix(y_test, predicted4)
```

In [122]:

```
print classification_report(y_test,predicted4)
```

```
             precision    recall  f1-score   support

          0       0.43      0.38      0.40         8
          1       0.57      0.73      0.64        11
          2       0.33      0.40      0.36         5
          3       0.00      0.00      0.00         2
          5       0.20      0.20      0.20         5
          6       1.00      0.50      0.67         2

avg / total       0.44      0.45      0.44        33
```

In [123]:

```
# Best score for KNN is after tuning of weights parameter- score=0.5625
```

In [124]:

```
from sklearn.utils import shuffle
new_Ind = []
cur_MaxScore = 0.0
col_num = 23
col_Ind_Random = shuffle(range(0,col_num), random_state=1)

for cur_f in range(0, col_num):
    new_Ind.append(col_Ind_Random[cur_f])
    newData = X.iloc[:, new_Ind]
    X_train, X_test, y_train, y_test = train_test_split(newData, y, test_size=0.2, rand
om_state=0)
    clf = KNeighborsClassifier(11, weights='distance')
    fit = clf.fit(X_train, y_train)
    cur_Score = clf.score(X_test, y_test)
    if cur_Score < cur_MaxScore:
        new_Ind.remove(col_Ind_Random[cur_f])
    else:
        cur_MaxScore = cur_Score
        print "Score with " + str(len(new_Ind)) + " selected features: " + str(cur_Scor
e)
```

```
Score with 1 selected features: 0.1282051282051282
Score with 2 selected features: 0.15384615384615385
Score with 3 selected features: 0.15384615384615385
Score with 4 selected features: 0.28205128205128205
Score with 5 selected features: 0.28205128205128205
Score with 6 selected features: 0.38461538461538464
Score with 7 selected features: 0.38461538461538464
Score with 8 selected features: 0.38461538461538464
Score with 9 selected features: 0.38461538461538464
Score with 10 selected features: 0.4358974358974359
Score with 11 selected features: 0.46153846153846156
Score with 12 selected features: 0.48717948717948717
Score with 13 selected features: 0.5128205128205128
Score with 14 selected features: 0.5384615384615384
Score with 15 selected features: 0.5384615384615384
Score with 16 selected features: 0.5384615384615384
Score with 17 selected features: 0.5384615384615384
Score with 18 selected features: 0.5384615384615384
Score with 19 selected features: 0.5384615384615384
```

In [125]:

```
print "There are " + str(len(new_Ind)) + " features selected:"
```

```
There are 19 features selected:
```

In [126]:

```
print new_Ind
```

```
[20, 17, 3, 13, 19, 16, 2, 6, 7, 1, 14, 0, 15, 22, 21, 9, 8, 12, 11]
```

In [127]:

```
X1= X.iloc[:, [0,1,2,3,6, 7, 8, 9, 11, 12,13, 14, 15,16, 17,19, 20, 21, 22]]
```

In [128]:

```
#X1=X.drop('landmass', axis=1)
#df.drop(columns=['B', 'C'])
#X1= X.drop(columns=['bars','colours','blue', 'white', 'crosses', 'saltires', 'quarter
s', 'animate', 'text'], axis=1)
```

In [129]:

```
X1.head(2)
```

Out[129]:

	bars	stripes	colours	red	gold	white	black	orange	crosses	saltires	quarters	suns
0	0	3	5	1	1	1	1	0	0	0	0	
1	0	0	3	1	1	0	1	0	0	0	0	

In [130]:

```
X1_train, X1_test, y_train, y_test = train_test_split(X1, y, test_size=0.20, random_sta
te=42, stratify=y) # stratified sampling
```

In [131]:

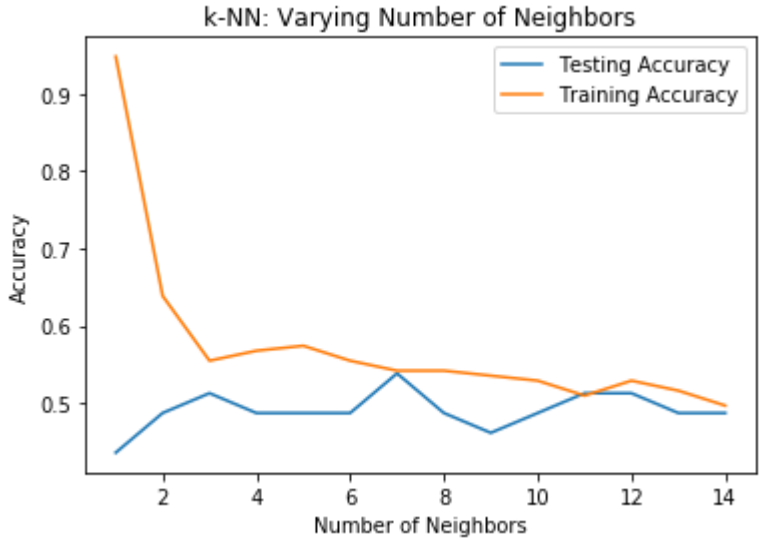
```
neighbors = np.arange(1, 15)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
for i, k in enumerate(neighbors):
    # Setup a k-NN Classifier with k neighbors: knn
    knn = KNeighborsClassifier(n_neighbors=k)

    # Fit the classifier to the training data
    knn.fit(X1_train, y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X1_train, y_train)

    #Compute accuracy on the testing set
    test_accuracy[i] = knn.score(X1_test, y_test)

# Generate plot
plt.title('k-NN: Varying Number of Neighbors')
plt.plot(neighbors, test_accuracy, label = 'Testing Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
```



In [ ]:

```
# Highest accuracy for K=6, 9,10,11,12,13,14
```

In [132]:

```
clf5 = KNeighborsClassifier(7)
```

In [133]:

```
clf5.fit(X1_train, y_train)
```

Out[133]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=7, p=2,
           weights='uniform')
```

In [134]:

```
predicted5 = clf5.predict(X1_test)
```

In [135]:

```
clf5.score(X1_test, y_test)
```

Out[135]:

```
0.5384615384615384
```

In [138]:

```
cm5 = confusion_matrix(y_test,predicted5)
```

In [139]:

```
print cm5
```

```
[[ 6  1  0  0  0  1  0  0]
 [ 0 12  0  0  0  0  0  0]
 [ 1  2  3  0  0  1  0  0]
 [ 0  1  0  0  0  1  0  0]
 [ 1  0  0  0  0  0  0  0]
 [ 2  2  1  0  0  0  0  0]
 [ 0  0  2  0  0  1  0  0]
 [ 0  0  0  0  0  1  0  0]]
```

In [140]:

```
print classification_report(y_test,predicted5)
```

```
             precision    recall  f1-score   support

          0       0.60      0.75      0.67         8
          1       0.67      1.00      0.80        12
          2       0.50      0.43      0.46         7
          3       0.00      0.00      0.00         2
          4       0.00      0.00      0.00         1
          5       0.00      0.00      0.00         5
          6       0.00      0.00      0.00         3
          7       0.00      0.00      0.00         1

avg / total       0.42      0.54      0.47        39
```

In [141]:

```
clf51 = KNeighborsClassifier(11, weights='distance')
```

In [142]:

```
fit = clf51.fit(X1_train, y_train)
```

In [143]:

```
predicted51 = fit.predict(X1_test)
```

In [144]:

```
clf51.score(X1_test, y_test)
```

Out[144]:

```
0.4358974358974359
```

In [145]:

```
cm51 = confusion_matrix(y_test, predicted51)
```

In [146]:

```
print cm51
```

```
[[5 1 1 0 0 0 1 0]
 [1 8 2 1 0 0 0 0]
 [1 2 3 0 0 1 0 0]
 [0 1 0 0 0 1 0 0]
 [1 0 0 0 0 0 0 0]
 [2 1 1 0 0 1 0 0]
 [0 0 2 0 0 1 0 0]
 [0 0 0 0 0 1 0 0]]
```

In [147]:

```
print classification_report(y_test,predicted51)
```

```
             precision    recall  f1-score   support

          0       0.50      0.62      0.56         8
          1       0.62      0.67      0.64        12
          2       0.33      0.43      0.38         7
          3       0.00      0.00      0.00         2
          4       0.00      0.00      0.00         1
          5       0.20      0.20      0.20         5
          6       0.00      0.00      0.00         3
          7       0.00      0.00      0.00         1

avg / total       0.38      0.44      0.40        39
```

Decision Tree

Without Hyper Parameters Tuning

In [148]:

```
#Without Hyper Parameters Tuning
#1-1,DesicionTree
#importing module
from sklearn.tree import DecisionTreeClassifier
#making the instance
model= DecisionTreeClassifier(random_state=1234)
#learning
model.fit(X_train,y_train)
#Prediction
prediction=model.predict(X_test)
#importing the metrics module
from sklearn import metrics
#evaluation(Accuracy)
print("Accuracy:",metrics.accuracy_score(prediction,y_test))
#evaluation(Confusion Metrix)
print("Confusion Metrix:\n",metrics.confusion_matrix(prediction,y_test))
```

```
('Accuracy:', 0.15384615384615385)
('Confusion Metrix:\n', array([[0, 4, 3, 1, 0, 1, 1, 0],
       [2, 3, 1, 0, 1, 1, 2, 0],
       [2, 3, 2, 0, 0, 1, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0],
       [2, 0, 0, 1, 0, 1, 0, 0],
       [1, 2, 1, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0]], dtype=int64))
```

With Hyper Parameters Tuning

In [149]:

```
#With Hyper Parameters Tuning
#2-1,DesicionTree
#importing modules
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
#making the instance
model= DecisionTreeClassifier(random_state=1234)
#Hyper Parameters Set
params = {'max_features': ['auto', 'sqrt', 'log2'],
          'min_samples_split': [2,3,4,5,6,7,8,9,10,11,12,13,14,15],
          'min_samples_leaf':[1,2,3,4,5,6,7,8,9,10,11],
          'random_state':[123]}
#Making models with hyper parameters sets
model1 = GridSearchCV(model, param_grid=params, n_jobs=-1)
#Learning
model1.fit(X_train,y_train)
#The best hyper parameters set
print("Best Hyper Parameters:",model1.best_params_)
#Prediction
prediction=model1.predict(X_test)
#importing the metrics module
from sklearn import metrics
#evaluation(Accuracy)
print("Accuracy:",metrics.accuracy_score(prediction,y_test))
#evaluation(Confusion Metrix)
print("Confusion Metrix:\n",metrics.confusion_matrix(prediction,y_test))
```

```
('Best Hyper Parameters:', {'max_features': 'auto', 'min_samples_split':
2, 'random_state': 123, 'min_samples_leaf': 10})
('Accuracy:', 0.2564102564102564)
('Confusion Metrix:\n', array([[0, 0, 1, 0, 0, 0, 0, 0],
       [5, 7, 5, 0, 1, 0, 1, 0],
       [2, 3, 1, 1, 0, 3, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [1, 2, 0, 1, 0, 2, 2, 1],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0]], dtype=int64))
```

Random Forest

Without Hyper Parameters Tuning

In [150]:

```
#Without Hyper Parameters Tuning
#1-2,Randomforest
#importing module
from sklearn.ensemble import RandomForestClassifier
#making the instance
model=RandomForestClassifier(n_jobs=-1,random_state=123)
#learning
model.fit(X_train,y_train)
#Prediction
prediction=model.predict(X_test)
#importing the metrics module
from sklearn import metrics
#evaluation(Accuracy)
print("Accuracy:",metrics.accuracy_score(prediction,y_test))
#evaluation(Confusion Metrix)
print("Confusion Metrix:\n",metrics.confusion_matrix(prediction,y_test))
```

```
('Accuracy:', 0.07692307692307693)
('Confusion Metrix:\n', array([[1, 2, 3, 0, 0, 0, 1, 1],
       [1, 1, 3, 1, 0, 2, 1, 0],
       [2, 3, 1, 0, 1, 2, 1, 0],
       [1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0],
       [2, 5, 0, 1, 0, 0, 0, 0],
       [1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0]], dtype=int64))
```

With Hyper Parameters Tuning

In [151]:

```
#With Hyper Parameters Tuning
#Randomforest
#importing modules
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
#making the instance
model=RandomForestClassifier()
#hyper parameters set
params = {'criterion':['gini','entropy'],
          'n_estimators':[10,15,20,25,30],
          'min_samples_leaf':[1,2,3],
          'min_samples_split':[3,4,5,6,7],
          'random_state':[123],
          'n_jobs':[-1]}
#Making models with hyper parameters sets
model1 = GridSearchCV(model, param_grid=params, n_jobs=-1)
#learning
model1.fit(X_train,y_train)
#The best hyper parameters set
print("Best Hyper Parameters:\n",model1.best_params_)
#Prediction
prediction=model1.predict(X_test)
#importing the metrics module
from sklearn import metrics
#evaluation(Accuracy)
print("Accuracy:",metrics.accuracy_score(prediction,y_test))
#evaluation(Confusion Metrix)
print("Confusion Metrix:\n",metrics.confusion_matrix(prediction,y_test))
```

```
('Best Hyper Parameters:\n', {'n_jobs': -1, 'min_samples_leaf': 2, 'n_esti
mators': 15, 'random_state': 123, 'criterion': 'entropy', 'min_samples_spl
it': 3})
('Accuracy:', 0.20512820512820512)
('Confusion Metrix:\n', array([[1, 0, 2, 1, 1, 2, 1, 0],
       [4, 6, 4, 0, 0, 0, 1, 1],
       [3, 5, 1, 0, 0, 2, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0],
       [0, 1, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0]], dtype=int64))
```

In [152]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [153]:

```
clf = DecisionTreeClassifier(random_state=0)
```

In [154]:

```
fit = clf.fit(X_train, y_train)
```

In [155]:

```
clf.score(X_test, y_test)
```

Out[155]:

```
0.1282051282051282
```

In [156]:

```
y_pre = fit.predict(X_test)
```

In [157]:

```
y_pre
```

Out[157]:

```
array([5, 0, 2, 2, 0, 6, 3, 0, 0, 6, 0, 2, 5, 6, 1, 2, 1, 2, 2, 2, 0, 1,
       0, 0, 1, 6, 2, 2, 0, 1, 0, 6, 6, 0, 2, 4, 1, 5, 2], dtype=int64)
```

In [158]:

```
y_pre.shape
```

Out[158]:

```
(39L,)
```

In [159]:

```
from sklearn.metrics import confusion_matrix
```

In [160]:

```
cm = confusion_matrix(y_test, y_pre)
```

In [161]:

```
print cm
```

```
[[0 1 3 1 0 1 2 0]
 [4 3 3 0 0 0 2 0]
 [3 1 2 0 0 0 1 0]
 [1 0 0 0 0 1 0 0]
 [0 0 1 0 0 0 0 0]
 [2 1 1 0 1 0 0 0]
 [1 0 1 0 0 1 0 0]
 [0 0 0 0 0 0 1 0]]
```

In [162]:

```
from sklearn.metrics import classification_report
```

In [163]:

```
print classification_report(y_test,y_pre)
```

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00         8
          1       0.50      0.25      0.33        12
          2       0.18      0.29      0.22         7
          3       0.00      0.00      0.00         2
          4       0.00      0.00      0.00         1
          5       0.00      0.00      0.00         5
          6       0.00      0.00      0.00         3
          7       0.00      0.00      0.00         1

avg / total       0.19      0.13      0.14        39
```

In [168]:

```
# Tree Visualization
```

In [169]:

```
from sklearn import tree
```

In [170]:

```
tree.export_graphviz(clf, out_file='tree.dot')
```

In [171]:

```
flags2 = X.iloc[:,0:22]
```

In [172]:

```
flags2.shape
```

Out[172]:

```
(194, 22)
```

In [173]:

```
feature_names =flags2.columns
```

In [174]:

```
feature_names
```

Out[174]:

```
Index([u'bars', u'stripes', u'colours', u'red', u'green', u'blue', u'gol
d',
       u'white', u'black', u'orange', u'circles', u'crosses', u'saltires',
       u'quarters', u'sunstars', u'crescent', u'triangle', u'icon', u'anim
ate',
       u'text', u'topleft_1', u'topleft_2'],
      dtype='object')
```

In [175]:

```
target_name= ['0','1','2', '3','4', '5', '6', '7']
```

In [176]:

```
try:
    import graphviz
except: import pip
    pip.main(['install','graphviz'])
```

```
  File "<ipython-input-176-106860b408dd>", line 4
    pip.main(['install','graphviz'])
    ^
IndentationError: unexpected indent
```

In [ ]:

```
import os
from sklearn import tree
os.environ["PATH"] += os.pathsep + "C:\\Users\\user\\Assignment 2"
from sklearn.tree import DecisionTreeClassifier,export_graphviz
import graphviz

data1 = export_graphviz(clf,out_file=None,feature_names=X.columns,class_names=target_na
me,
                        filled=True, rounded=True,
                        special_characters=True)
graph = graphviz.Source(data1)
graph
```

In [ ]:

```
with open('flags2.dot', 'w') as f:
    f = tree.export_graphviz(clf, out_file=f, feature_names= feature_names, class_names
=target_name, filled=True, rounded=True, special_characters=True)
```

In [ ]:

```

```

In [ ]:

```

```