In [2]:
```
import glob
import pandas as pd
from dbfread import DBF
# read in ROI dbf
Input_Raster_dir = 'D:/Box Sync/research/Leslie/Classification_sUAS/'
Files = glob.glob(Input_Raster_dir + '/**/Training*.dbf', recursive=True)
# Set the output dir
Output_dir = 'D:/Box Sync/research/Leslie/Classification_sUAS/Results'
```

In [3]:
```
# Read in training data table and display first 5 rows
features = pd.DataFrame()#creates a new dataframe that's empty
for file in Files:
    table = DBF(file)
    features_temp = pd.DataFrame(iter(table))
    features_temp.columns = ['Id','Class','ClassID','Blue','Green','Red','NIR'
,'Rededge','VegHeight','NDVI_April','NDVI_May','NDVI_June','sUAS_NDVI']
    features = pd.concat([features, features_temp])
```

In [4]:
```
print('The shape of our features is:', features.shape)
```

```
The shape of our features is: (6014, 13)
```

In [5]:
```
# Descriptive statistics for each column
features.describe()
```

Out[5]:

	Id	ClassID	Blue	Green	Red	NIR	Rec
count	6014.0	6014.000000	6014.000000	6014.000000	6014.000000	6014.000000	6014.0(
mean	0.0	4.091952	0.056439	0.080925	0.083267	0.146810	0.1909!
std	0.0	1.905560	0.025642	0.033232	0.059018	0.059349	0.1068
min	0.0	1.000000	0.009994	0.012880	0.002409	0.024032	0.0075
25%	0.0	3.000000	0.039164	0.063471	0.026308	0.114436	0.1071
50%	0.0	4.000000	0.053345	0.081959	0.077631	0.150257	0.2004
75%	0.0	6.000000	0.072944	0.100389	0.126918	0.183214	0.2774
max	0.0	7.000000	0.163086	0.218475	0.326172	0.392057	0.4587!

In [6]:
```
# Use numpy to convert to arrays
import numpy as np
# Labels are the values we want to predict
labels = np.array(features['ClassID'])
# Remove the labels from the features
# axis 1 refers to the columns
features= features.drop('Class', axis = 1)
#features= features.drop('FID', axis = 1)
features= features.drop('ClassID', axis = 1)
features= features.drop('Id', axis = 1)
# Saving feature names for later use
feature_list = list(features.columns)
# Convert to numpy array
print(features)
features = np.array(features)
```

	Blue	Green	Red	NIR	Rededge	VegHeight	NDVI_April
\							
0	0.091952	0.106185	0.143937	0.157177	0.192628	0.126385	0.228142
1	0.087162	0.095235	0.122546	0.136576	0.171924	0.850425	0.226555
2	0.093574	0.101496	0.128537	0.140148	0.174513	1.065720	0.226926
3	0.111606	0.118697	0.152314	0.166813	0.197587	0.687303	0.226372
4	0.046005	0.058446	0.076165	0.093235	0.136567	0.626777	0.229995
5	0.046587	0.054017	0.076584	0.091399	0.142014	0.033571	0.250942
6	0.056383	0.065289	0.096215	0.112271	0.157856	0.011269	0.249963
7	0.067594	0.078522	0.104881	0.116195	0.160354	0.040925	0.252883
8	0.064289	0.072396	0.104702	0.123317	0.165908	0.078159	0.257092
9	0.086794	0.099210	0.134296	0.150704	0.186256	0.025253	0.229462
10	0.090335	0.107933	0.150371	0.161399	0.197377	0.068544	0.226694
11	0.077091	0.077849	0.114429	0.128694	0.174113	0.040858	0.230302
12	0.083846	0.092669	0.129561	0.144306	0.184619	0.107741	0.237923
13	0.078443	0.083778	0.113965	0.128084	0.174708	0.108935	0.242691
14	0.070653	0.081463	0.117598	0.134939	0.175468	0.058397	0.224602
15	0.069440	0.083813	0.122599	0.139101	0.178876	0.134508	0.228176
16	0.068167	0.075076	0.104495	0.122086	0.167760	0.008479	0.231836
17	0.076297	0.087480	0.122199	0.135295	0.178386	-0.000558	0.236126
18	0.060629	0.068419	0.101020	0.118785	0.165917	0.018175	0.235375
19	0.070901	0.080961	0.118128	0.137348	0.180797	0.004288	0.237137
20	0.075148	0.085487	0.109639	0.129603	0.179441	-0.000581	0.241854
21	0.058655	0.065730	0.096268	0.113718	0.161971	-0.005839	0.229914
22	0.073175	0.091140	0.121271	0.137399	0.176166	0.056616	0.229844
23	0.075855	0.095944	0.133211	0.158446	0.200586	0.051458	0.230829
24	0.083321	0.091210	0.113456	0.125631	0.160006	0.043084	0.228302
25	0.064458	0.072292	0.091542	0.110209	0.149916	0.047850	0.223303
26	0.084982	0.095265	0.120881	0.136704	0.169405	0.029937	0.218619
27	0.082812	0.100797	0.127717	0.144466	0.172749	0.067080	0.222066
28	0.078734	0.097110	0.128795	0.150850	0.185873	-0.000385	0.228260
29	0.083099	0.097013	0.128452	0.141069	0.175298	0.046215	0.233057
..	...	...	...	...	...	...	...
329	0.042599	0.100619	0.065976	0.182419	0.368554	5.350600	0.268042
330	0.039824	0.087071	0.067257	0.165899	0.330894	4.839810	0.285457
331	0.041664	0.078978	0.074687	0.138645	0.257494	9.759470	0.343435
332	0.041533	0.089877	0.073414	0.178054	0.327755	4.358700	0.353215
333	0.037123	0.081189	0.066155	0.155172	0.308555	6.309070	0.325425
334	0.034958	0.084278	0.060901	0.152241	0.297946	6.659710	0.308300
335	0.032594	0.070057	0.055976	0.137083	0.284414	7.219080	0.318037
336	0.035624	0.088624	0.059507	0.166651	0.344897	5.620740	0.305641
337	0.037497	0.083686	0.074283	0.165329	0.308685	5.217090	0.329905
338	0.038252	0.090741	0.066736	0.176680	0.359560	4.519570	0.326198
339	0.031195	0.064422	0.054088	0.127782	0.307112	5.029460	0.329340
340	0.021819	0.038453	0.043127	0.065118	0.155479	4.804300	0.329187
341	0.022234	0.051386	0.036092	0.093224	0.190642	6.555410	0.315443
342	0.037517	0.093939	0.064217	0.181292	0.311109	6.302570	0.314337
343	0.029417	0.067833	0.054157	0.144152	0.269676	6.289150	0.294465
344	0.047671	0.106316	0.084505	0.192503	0.337494	7.884720	0.317862
345	0.044698	0.104829	0.079163	0.191636	0.348396	6.715180	0.317115
346	0.032213	0.075342	0.056084	0.142399	0.300414	7.986700	0.304377
347	0.028335	0.063168	0.045827	0.124260	0.272439	7.469810	0.298329
348	0.037281	0.089098	0.061211	0.177658	0.373092	5.222680	0.296156
349	0.033598	0.073994	0.056558	0.144144	0.300179	5.671680	0.280994
350	0.024992	0.053865	0.037441	0.108631	0.233736	7.276960	0.313180
351	0.033687	0.077478	0.054609	0.149144	0.316916	6.030150	0.317156
352	0.037503	0.087738	0.061132	0.171843	0.364320	7.902660	0.310266

353	0.033221	0.070694	0.054710	0.145651	0.308018	4.701450	0.291765
354	0.038065	0.081931	0.063062	0.162733	0.351590	6.355900	0.311877
355	0.045338	0.098638	0.085138	0.193697	0.354455	4.524310	0.319068
356	0.032325	0.062930	0.054628	0.124147	0.273131	7.049620	0.302768
357	0.035895	0.079101	0.055607	0.141340	0.267583	5.974810	0.229576
358	0.043255	0.093521	0.073459	0.169365	0.330367	4.940590	0.255928

	NDVI_May	NDVI_June	sUAS_NDVI
0	0.246652	0.165636	0.144671
1	0.249433	0.165185	0.167684
2	0.250859	0.168021	0.151710
3	0.253833	0.167397	0.129389
4	0.256580	0.172457	0.283931
5	0.263574	0.180530	0.299314
6	0.258150	0.177170	0.242613
7	0.264681	0.190565	0.209148
8	0.258677	0.203633	0.226178
9	0.248969	0.203753	0.162094
10	0.251777	0.206075	0.135174
11	0.253287	0.205197	0.206846
12	0.252855	0.204314	0.175244
13	0.252341	0.202300	0.210421
14	0.252208	0.200156	0.197461
15	0.251332	0.200598	0.186674
16	0.251502	0.203269	0.232374
17	0.251046	0.204207	0.186925
18	0.249854	0.197451	0.243114
19	0.250320	0.195437	0.209650
20	0.253076	0.205159	0.241465
21	0.254508	0.205500	0.254428
22	0.251222	0.200171	0.184560
23	0.248850	0.198312	0.201843
24	0.265643	0.191800	0.170223
25	0.266896	0.193434	0.241758
26	0.269508	0.197897	0.167157
27	0.270846	0.201703	0.149875
28	0.274478	0.205204	0.181389
29	0.265267	0.190228	0.154225
..	...	...	...
329	0.298976	0.224788	0.696334
330	0.314521	0.276586	0.662153
331	0.296427	0.280124	0.550324
332	0.304863	0.253801	0.633999
333	0.303846	0.282766	0.646899
334	0.306576	0.253504	0.660572
335	0.315979	0.263805	0.671107
336	0.316286	0.269575	0.705704
337	0.284264	0.261767	0.612065
338	0.285302	0.276086	0.686902
339	0.284688	0.297278	0.700509
340	0.269985	0.258530	0.565704
341	0.300634	0.270469	0.681636
342	0.308444	0.283290	0.657808
343	0.299573	0.271500	0.665526
344	0.327260	0.302796	0.599501
345	0.332773	0.299824	0.629697
346	0.324602	0.309611	0.685364

```
347  0.318094   0.290259   0.712021
348  0.330327   0.286220   0.718119
349  0.293143   0.243830   0.682915
350  0.324227   0.282492   0.723861
351  0.299686   0.275577   0.706025
352  0.314875   0.303845   0.712627
353  0.313849   0.288269   0.698344
354  0.326974   0.270511   0.695831
355  0.332953   0.337898   0.612652
356  0.324060   0.339915   0.666659
357  0.286782   0.271768   0.655888
358  0.267662   0.248193   0.636186

[6014 rows x 10 columns]
```

In [7]:
```
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_siz
e = 0.25, random_state = 40)
```

In [8]:
```
print('Training Features Shape:', X_train.shape)
print('Training Labels Shape:', y_train.shape)
print('Testing Features Shape:', X_test.shape)
print('Testing Labels Shape:', y_test.shape)
```

```
Training Features Shape: (4510, 10)
Training Labels Shape: (4510,)
Testing Features Shape: (1504, 10)
Testing Labels Shape: (1504,)
```

In [9]:
```
from sklearn import model_selection
# Import the model we are using
from sklearn.ensemble import RandomForestClassifier
# random forest model creation
rfc = RandomForestClassifier(oob_score=True)
rfc.fit(X_train,y_train)
# predictions
rfc_predict = rfc.predict(X_test)
```

```
C:\Users\GraceLiu\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:453:
UserWarning: Some inputs do not have OOB scores. This probably means too few
trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
C:\Users\GraceLiu\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:458:
RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

In [10]:
```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
import sklearn
y = sklearn.preprocessing.label_binarize(labels, classes=[1, 2, 3, 4, 5])
X = features
#rfc_cv_score = cross_val_score(rfc,X, y, cv=2, scoring='roc_auc')
print("=== Confusion Matrix ===")
print(confusion_matrix(y_test, rfc_predict))
print('\n')
print("=== Classification Report ===")
print(classification_report(y_test, rfc_predict))
print('\n')
# print("=== All AUC Scores ===")
# print(rfc_cv_score)
# print('\n')
# print("=== Mean AUC Score ===")
# print("Mean AUC Score - Random Forest: ", rfc_cv_score.mean())
print('Our OOB prediction of accuracy is: {oob}%'.format(oob=rfc.oob_score_ *
100))
```

```
=== Confusion Matrix ===
[[170   4   5   0   0   3   0]
 [  6 149   7   3   1  13   1]
 [  4   0 268   2   0   0   0]
 [  0   3   2 188   0   4   3]
 [  0   0   0   0 274   1   3]
 [  6   8   2   3   0 149   1]
 [  2   4   0   7   1   2 205]]


=== Classification Report ===
             precision    recall  f1-score   support

          1       0.90      0.93      0.92       182
          2       0.89      0.83      0.86       180
          3       0.94      0.98      0.96       274
          4       0.93      0.94      0.93       200
          5       0.99      0.99      0.99       278
          6       0.87      0.88      0.87       169
          7       0.96      0.93      0.94       221

avg / total       0.93      0.93      0.93      1504



Our OOB prediction of accuracy is: 89.95565410199556%
```

In [11]:

```
from sklearn.model_selection import RandomizedSearchCV
# number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10
)]
# number of features at every split
max_features = ['auto', 'sqrt']

# max depth
max_depth = [int(x) for x in np.linspace(100, 500, num = 11)]
max_depth.append(None)
# create random grid
random_grid = {
 'n_estimators': n_estimators,
 'max_features': max_features,
 'max_depth': max_depth
 }
# Random search of parameters
rfc_random = RandomizedSearchCV(estimator = rfc, param_distributions = random_
grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the model
rfc_random.fit(X_train, y_train)
# print results
print(rfc_random.best_params_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  6.5min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 14.9min finished

{'n_estimators': 1400, 'max_features': 'auto', 'max_depth': 260}
```

In [14]:
```
rfc = RandomForestClassifier(n_estimators = 1400,#rfc_random.best_params_['n_e
stimators'],
                             max_features = 'auto',#rfc_random.best_params_['m
ax_features'],
                             max_depth = 260,#rfc_random.best_params_['max_dep
th'],
                             oob_score=True)
rfc.fit(X_train,y_train)
rfc_predict = rfc.predict(X_test)
rfc_predict_train = rfc.predict(X_train)
print("=== Confusion Matrix (test) ===")
print(confusion_matrix(y_test, rfc_predict))
print('\n')
print("=== Confusion Matrix (train) ===")
print(confusion_matrix(y_train, rfc_predict_train))
print('\n')
print("=== Classification Report ===")
print(classification_report(y_test, rfc_predict))
print('\n')
print('Our OOB prediction of accuracy is: {oob}%'.format(oob=rfc.oob_score_ *
100))
```

```
=== Confusion Matrix (test) ===
[[172   3   4   0   0   3   0]
 [  4 154   4   4   1  12   1]
 [  1   1 270   2   0   0   0]
 [  0   3   1 188   1   4   3]
 [  0   0   0   0 275   1   2]
 [  5   9   0   3   0 151   1]
 [  0   4   1   4   1   3 208]]


=== Confusion Matrix (train) ===
[[511   0   0   0   0   0   0]
 [  0 514   0   0   0   0   0]
 [  0   0 841   0   0   0   0]
 [  0   0   0 617   0   0   0]
 [  0   0   0   0 848   0   0]
 [  0   0   0   0   0 529   0]
 [  0   0   0   0   0   0 650]]


=== Classification Report ===
             precision    recall  f1-score   support

          1       0.95      0.95      0.95       182
          2       0.89      0.86      0.87       180
          3       0.96      0.99      0.97       274
          4       0.94      0.94      0.94       200
          5       0.99      0.99      0.99       278
          6       0.87      0.89      0.88       169
          7       0.97      0.94      0.95       221

avg / total       0.94      0.94      0.94      1504



Our OOB prediction of accuracy is: 94.7450110864745%
```

In [15]:

```
# Fit the model
rfc_random.fit(features, labels)
# print results
print(rfc_random.best_params_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   58.8s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  5.6min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 11.5min finished

{'n_estimators': 2000, 'max_features': 'auto', 'max_depth': 300}
```

In [16]:
```
rfc_full = RandomForestClassifier(n_estimators = 2000,#rfc_random.best_params_
['n_estimators'],
                                  max_features ='auto',# rfc_random.best_param
s_['max_features'],
                                  max_depth = 300,#rfc_random.best_params_['ma
x_depth'],
                                  oob_score=True)
rfc_full.fit(features,labels)
print('Our OOB prediction of accuracy for the full model is: {oob}%'.format(oo
b=rfc_full.oob_score_ * 100))
```

```
Our OOB prediction of accuracy for the full model is: 95.12803458596608%
```

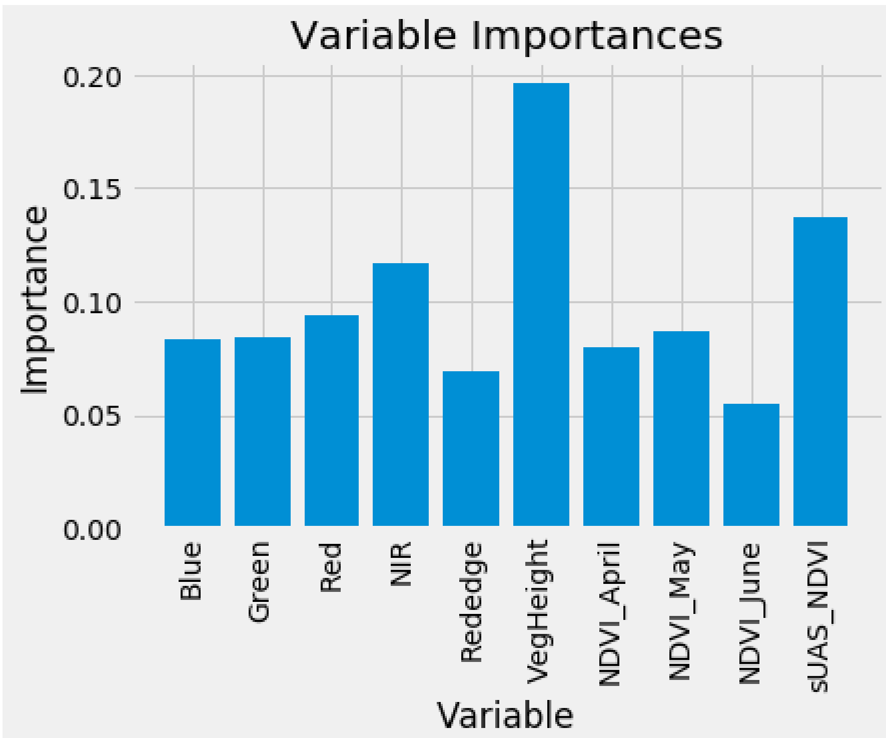
In [15]:
```
import os
# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot
# Pull out one tree from the forest
tree = rfc.estimators_[5]
# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot
# Pull out one tree from the forest
tree = rfc.estimators_[5]
# Export the image to a dot file
export_graphviz(tree, out_file = 'Tree.dot', feature_names = feature_list, rou
nded = True, precision = 1)
# Use dot file to create a graph
(graph, ) = pydot.graph_from_dot_file('Tree.dot')
# Write graph to a png file
graph.write_png(os.path.join(Output_dir,'Tree.tif'))
```

In [19]:
```
# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot
# Pull out one tree from the forest
tree = rfc_full.estimators_[5]
# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot
# Pull out one tree from the forest
tree = rfc_full.estimators_[5]
# Export the image to a dot file
export_graphviz(tree, out_file = 'Tree_full.dot', feature_names = feature_list
, rounded = True, precision = 1)
# Use dot file to create a graph
(graph, ) = pydot.graph_from_dot_file('Tree_full.dot')
# Write graph to a png file
graph.write_png(os.path.join(Output_dir,'Tree_full.png'))
```

In [17]:
```
# Get numerical feature importances
importances = list(rfc.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance
in zip(feature_list, importances)]
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], revers
e = True)
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_imp
ortances];
```

```
Variable: VegHeight            Importance: 0.2
Variable: sUAS_NDVI            Importance: 0.14
Variable: NIR                  Importance: 0.12
Variable: Red                  Importance: 0.09
Variable: NDVI_May             Importance: 0.09
Variable: Blue                 Importance: 0.08
Variable: Green                Importance: 0.08
Variable: NDVI_April           Importance: 0.08
Variable: Rededge              Importance: 0.07
Variable: NDVI_June            Importance: 0.05
```

In [18]:
```
# Import matplotlib for plotting and use magic command for Jupyter Notebooks
import matplotlib.pyplot as plt
%matplotlib inline
# Set the style
plt.style.use('fivethirtyeight')
# list of x locations for plotting
x_values = list(range(len(importances)))
# Make a bar chart
plt.bar(x_values, importances, orientation = 'vertical')
# Tick labels for x axis
plt.xticks(x_values, feature_list, rotation='vertical')
# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Variable Importan
ces');
```



In [26]:
```
# # saving full model
# from sklearn.externals import joblib
# # Save to file in the current working directory
# joblib_file = os.path.join(Output_dir,'RFmodel_10b_full.pickle')
# joblib.dump(rfc_full, joblib_file)

# # Load from file
# joblib_model = joblib.load(joblib_file)

# # Calculate the accuracy and predictions
# score = joblib_model.score(X_test, y_test)
# print("Test score: {0:.2f} %".format(100 * score))
```

```
Test score: 100.00 %
```

In [20]:
```
# saving full model using pickle
import os
import pickle
# Save to file in the current working directory
pkl_filename = os.path.join(Output_dir,'RFmodel_10b_full_V2.pickle')
with open(pkl_filename, 'wb') as file:
    pickle.dump(rfc_full, file)
# # Load from file
# with open(pkl_filename, 'rb') as file:
#     pickle_model = pickle.load(file)

# # Calculate the accuracy score and predict target values
# score = pickle_model.score(X_test, y_test)
# print("Test score: {0:.2f} %".format(100 * score))
```

In [21]:
```
# Predicting the rest of the image
from osgeo import gdal, gdal_array
import matplotlib.pyplot as plt
import os
import osr
def raster2array(rasterfn,i):
    raster = gdal.Open(rasterfn)
    band = raster.GetRasterBand(i)
    return band.ReadAsArray()
def array2raster(rasterfn,newRasterfn,array):
    raster = gdal.Open(rasterfn)
    geotransform = raster.GetGeoTransform()
    originX = geotransform[0]
    originY = geotransform[3]
    pixelWidth = geotransform[1]
    pixelHeight = geotransform[5]
    cols = raster.RasterXSize
    rows = raster.RasterYSize
    driver = gdal.GetDriverByName('GTiff')
    outRaster = driver.Create(newRasterfn, cols, rows, 1, gdal.GDT_Float32)
    outRaster.SetGeoTransform((originX, pixelWidth, 0, originY, 0, pixelHeight
))
    outband = outRaster.GetRasterBand(1)
    outband.WriteArray(array)
    outRasterSRS = osr.SpatialReference()
    outRasterSRS.ImportFromWkt(raster.GetProjectionRef())
    outRaster.SetProjection(outRasterSRS.ExportToWkt())
```

In [22]:
```
# Read in training data table and display first 5 rows
#Sites = ['Rush_Fire']#,'Ashley_Fire','Blue_Door_Fire','Blue_Fire','Horse_Fire
_North','Horse_Fire_South','Horse_Lake_Fire','Nelson_Fire','Scorpion_Fire']
Files = glob.glob(Input_Raster_dir + '/**/Input_*.tif', recursive=True)
for file in Files:
    print('working on ' + file )
    site = os.path.basename(file)[6:-4]
    img_ds = gdal.Open(file, gdal.GA_ReadOnly)
    img = np.zeros((img_ds.RasterYSize, img_ds.RasterXSize, img_ds.RasterCount
),
                   gdal_array.GDALTypeCodeToNumericTypeCode(img_ds.GetRasterBa
nd(1).DataType))
    for i in np.arange(img_ds.RasterCount):
        img[:,:,i] = raster2array(file,int(i)+1)
    # Take our full image, ignore the Fmask band, and reshape into long 2d arr
ay (nrow * ncol, nband) for classification
    new_shape = (img.shape[0] * img.shape[1], img.shape[2])
    img_as_array = img[:, :, :].reshape(new_shape)
    print('Reshaped from {o} to {n}'.format(o=img.shape,
                                            n=img_as_array.shape))
    # Now predict for each pixel
    class_prediction = rfc_full.predict(img_as_array)
    # Reshape our classification map
    class_prediction = class_prediction.reshape(img[:, :, 0].shape)
    # Set example dir for meta data
    Example_dir = glob.glob(os.path.join(Input_Raster_dir,site,'Raster','I_Cli
ped','*Multispectral_NDVI.tif'))[0]
    # Set the output dir
    Classification_dir = os.path.join(Input_Raster_dir,site,'Raster','III_Clas
sification')
    if not os.path.exists(Classification_dir):
        os.makedirs(Classification_dir)
    array2raster(Example_dir,
                os.path.join(Classification_dir,site+'_RFClass_V2.tif'),
                class_prediction)
```

```
working on D:/Box Sync/research/Leslie/Classification_sUAS\Ashley_Fire\Raster
\II_StackedInput\Input_Ashley_Fire.tif
Reshaped from (5491, 6673, 10) to (36641443, 10)

C:\Users\GraceLiu\Anaconda3\lib\site-packages\numpy\core\_methods.py:32: Runt
imeWarning: overflow encountered in reduce
  return umr_sum(a, axis, dtype, out, keepdims)

working on D:/Box Sync/research/Leslie/Classification_sUAS\Blue_Door_Fire\Ras
ter\II_StackedInput\Input_Blue_Door_Fire.tif
Reshaped from (5577, 4548, 10) to (25364196, 10)
working on D:/Box Sync/research/Leslie/Classification_sUAS\Blue_Fire\Raster\I
I_StackedInput\Input_Blue_Fire.tif
Reshaped from (4084, 5653, 10) to (23086852, 10)
working on D:/Box Sync/research/Leslie/Classification_sUAS\Horse_Fire_North\R
aster\II_StackedInput\Input_Horse_Fire_North.tif
Reshaped from (5840, 7601, 10) to (44389840, 10)
working on D:/Box Sync/research/Leslie/Classification_sUAS\Horse_Fire_South\R
aster\II_StackedInput\Input_Horse_Fire_South.tif
Reshaped from (5932, 7105, 10) to (42146860, 10)
working on D:/Box Sync/research/Leslie/Classification_sUAS\Horse_Lake_Fire\Ra
ster\II_StackedInput\Input_Horse_Lake_Fire.tif
Reshaped from (6758, 4244, 10) to (28680952, 10)
working on D:/Box Sync/research/Leslie/Classification_sUAS\Nelson_Fire\Raster
\II_StackedInput\Input_Nelson_Fire.tif
Reshaped from (6805, 6375, 10) to (43381875, 10)
working on D:/Box Sync/research/Leslie/Classification_sUAS\Rush_Fire\Raster\I
I_StackedInput\Input_Rush_Fire.tif
Reshaped from (4868, 5671, 10) to (27606428, 10)
working on D:/Box Sync/research/Leslie/Classification_sUAS\Scorpion_Fire\Rast
er\II_StackedInput\Input_Scorpion_Fire.tif
Reshaped from (4786, 8005, 10) to (38311930, 10)
```

In [23]:

```
## visiualization
# plt.imshow(class_prediction, interpolation='none')
# plt.show()
```