Random Forest prediction of potential fire control locations (PCLs) for the 2018 Polecreek Fire, Utah (presented at the 2019 International Fire Ecology and Management Conference)

Cite as: Hallema, D. W., O'Connor, C. J., Thompson, M. P., Sun, G., McNulty, S. G., Calkin, D. E. & Martin, K. L. (2019). Predicting fire line effectiveness with machine learning. 8th International Fire Ecology and Management Conference, *Association for Fire Ecology*, Tucson, Arizona, November 18-22, 2019.

Author: [Dennis W. Hallema (https://www.linkedin.com/in/dennishallema)](https://www.linkedin.com/in/dennishallema)

Description: Random Forest prediction of potential fire control locations (PCLs) for the 2018 Polecreek Fire in Utah. Prediction of PCLs is key to effective pre-fire planning and fire operations management.

Depends: See `environment.yml` .

Data: Topography, fuel characteristics, road networks and fire suppression

Cite as: Hallema, D. W., O'Connor, C. J., Thompson, M. P., Sun, G., McNulty, S. G., Calkin, D. E. & Martin, K. L. (2019). Predicting fire line effectiveness with machine learning. 8th International Fire Ecology and Management Conference, *Association for Fire Ecology*, Tucson, Arizona, November 18-22, 2019.

Acknowledgement: Funding was provided by the USDA Forest Service Rocky Mountain Research Station through an agreement between the USDA Forest Service Southern Research Station and North Carolina State University (agreement number 19-CS-11330110-075).

Content:

- Data preparation
- Random Forest (RF) classification
- Feature importance
- Classifier optimization

Data preparation

In [1]:
```
# Import modules
import numpy as np
import matplotlib.colors as colors
import matplotlib.pyplot as plt
%matplotlib inline
from osgeo import gdal, gdal_array
from sklearn.metrics import confusion_matrix, classification_report, mean_square
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import Binarizer, OrdinalEncoder
gdal.UseExceptions()
gdal.AllRegister()
```

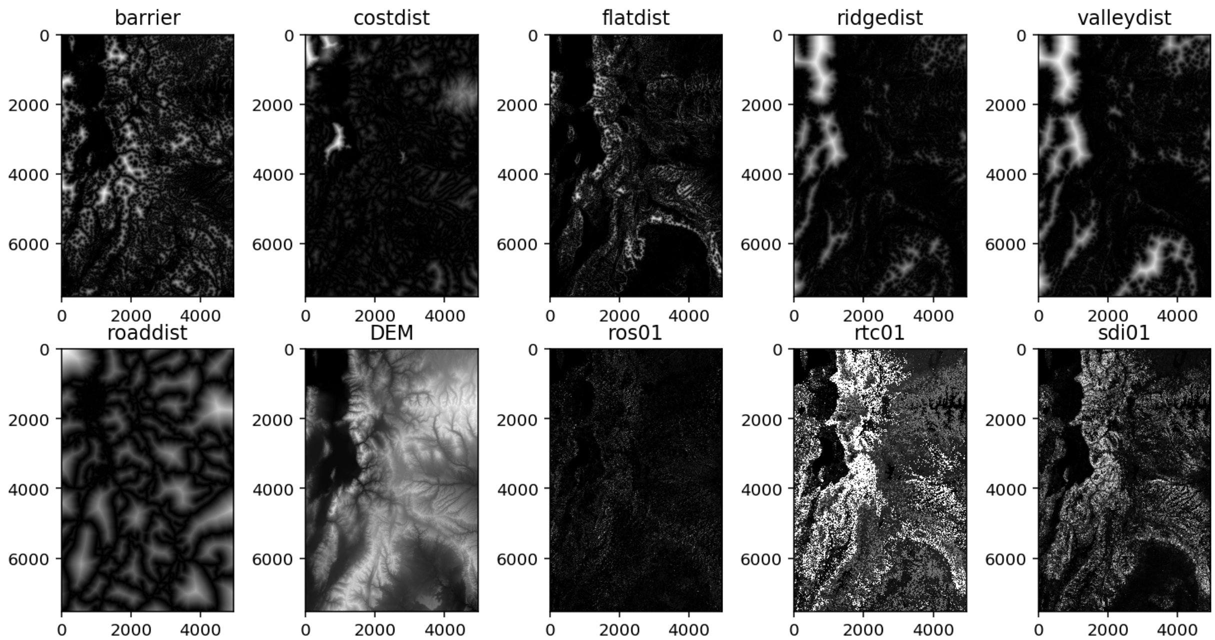
In [2]:
```
# Raster input files
features = [
    'data/barrier.tif',
    'data/costdist.tif',
    'data/flatdist.tif',
    'data/ridgedist.tif',
    'data/valleydist.tif',
    'data/roaddist.tif',
    'data/DEM.tif',
    'data/ros01.tif',
    'data/rtc01.tif',
    'data/sdi01.tif'
]
response = ['data/brt_resp2.tif']
```

In [3]:
```
# Create data labels
feature_list = [str.split(features[i],"/")[-1] for i in range(len(features))]
feature_list = [str.split(feature_list[i],".")[-2] for i in range(len(features))
response_list = [str.split(response[i],"/")[-1] for i in range(len(response))]
response_list = [str.split(response_list[i],".")[-2] for i in range(len(response
```

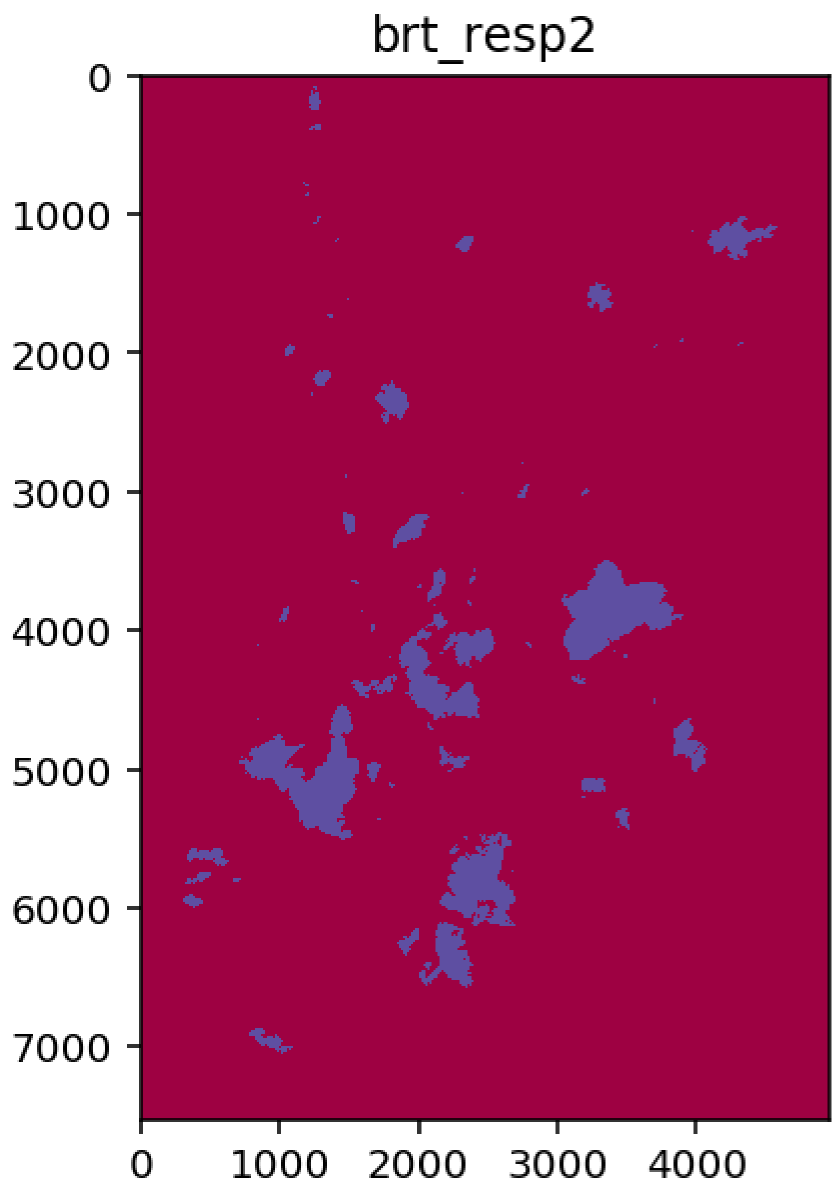
In [4]:
```
# Read response data
ras_ds = gdal.Open(response[0], gdal.GA_ReadOnly)
y = ras_ds.GetRasterBand(1).ReadAsArray()

# Read feature data
X = np.zeros((ras_ds.RasterYSize, ras_ds.RasterXSize, len(features)), dtype=floa
for b, f in enumerate(features):
    ras_ds = gdal.Open(f, gdal.GA_ReadOnly)
    X[:,:,b] = ras_ds.GetRasterBand(1).ReadAsArray()

print("Feature array dimensions: {}".format(X.shape))
print("Response array dimensions: {}".format(y.shape))
```

```
Feature array dimensions: (7525, 4960, 10)
Response array dimensions: (7525, 4960)
```

In [5]:
```python
# Plot feature maps
plt.rcParams['figure.figsize'] = [12.8, 6.4]
plt.rcParams['figure.dpi'] = 144
fig, axes = plt.subplots(2,5)

for i, ax in zip(range(X.shape[2]), axes.flatten()):
    ax.imshow(X[:,:,i], cmap=plt.cm.Greys_r)
    ax.set_title(str(feature_list[i]))
```



In [6]:
```
# Plot response map
plt.rcParams['figure.figsize'] = [6.4, 4.8]
plt.rcParams['figure.dpi'] = 144
plt.imshow(y, cmap=plt.cm.Spectral)
plt.title(str(response_list[0]))
plt.show()
```



In [7]:
```
# Apply mask
mask = Binarizer(threshold = -0.000001).fit_transform(y)
for i in range(X.shape[2]):
    X[:,:,i] = X[:,:,i] * mask
```

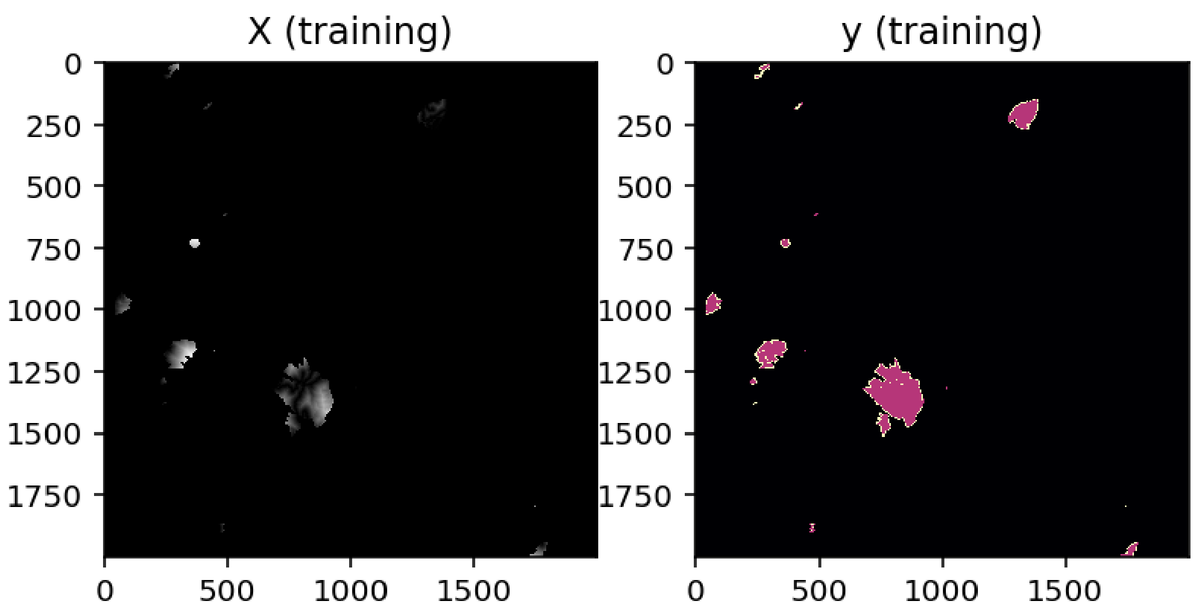
In [8]:
```
# Subset training and testing maps
X_train = X[1000:3000, 1000:3000, 0:X.shape[2]]
y_train = y[1000:3000, 1000:3000]
X_test = X[3000:4000, 1000:3000, 0:X.shape[2]]
y_test = y[3000:4000, 1000:3000]

print("X_train shape {}".format(X_train.shape))
print("y_train shape {}".format(y_train.shape))
print("X_test shape {}".format(X_test.shape))
print("y_test shape {}".format(y_test.shape))
```

```
X_train shape (2000, 2000, 10)
y_train shape (2000, 2000)
X_test shape (1000, 2000, 10)
y_test shape (1000, 2000)
```

In [9]:
```
# Encode response arrays
y_train = OrdinalEncoder().fit_transform(y_train)
y_test = OrdinalEncoder().fit_transform(y_test)
```

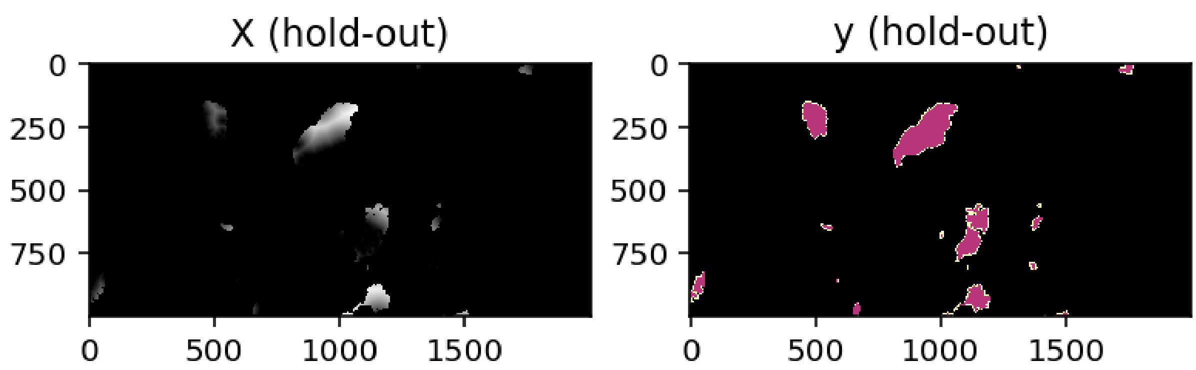
In [10]:

```
# Plot training map
plt.rcParams['figure.figsize'] = [6.4, 4.8]
plt.rcParams['figure.dpi'] = 144

plt.subplot(121)
plt.imshow(X_train[:,:,1], cmap=plt.cm.Greys_r)
plt.title('X (training)')

plt.subplot(122)
plt.imshow(y_train, cmap=plt.cm.get_cmap('magma'))
plt.title('y (training)')

plt.show()
```



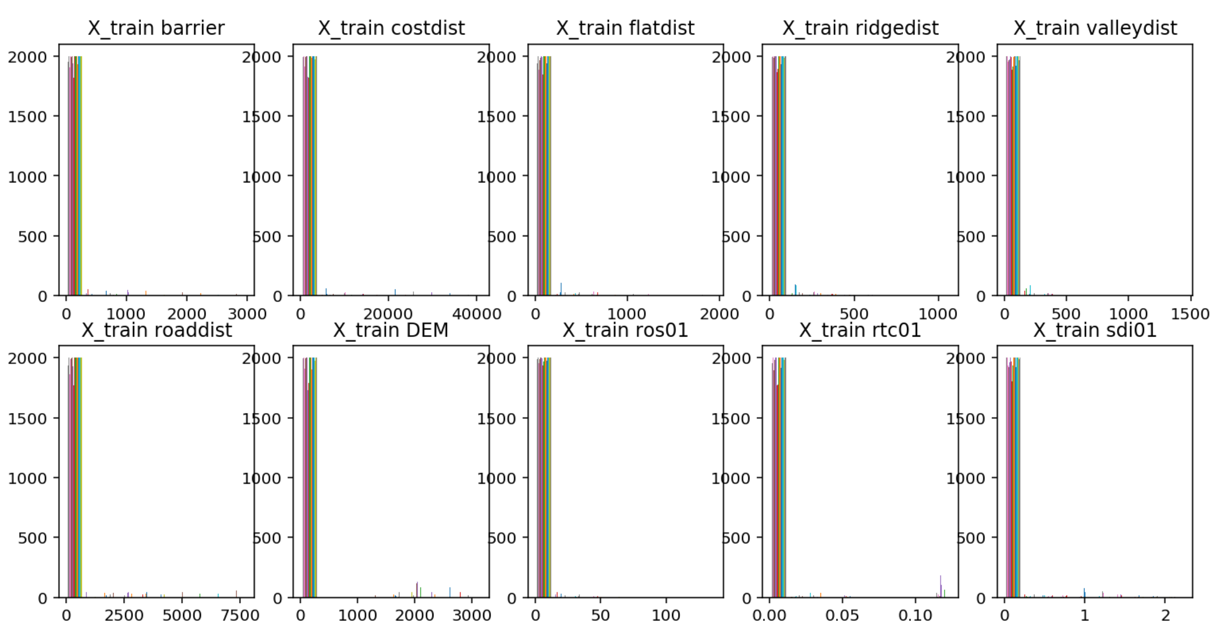
In [11]:

```python
# Plot testing map
plt.rcParams['figure.figsize'] = [6.4, 4.8]
plt.rcParams['figure.dpi'] = 144

plt.subplot(121)
plt.imshow(X_test[:,:,0], cmap=plt.cm.Greys_r)
plt.title('X (hold-out)')

plt.subplot(122)
plt.imshow(y_test, cmap=plt.cm.get_cmap('magma'))
plt.title('y (hold-out)')

plt.show()
```

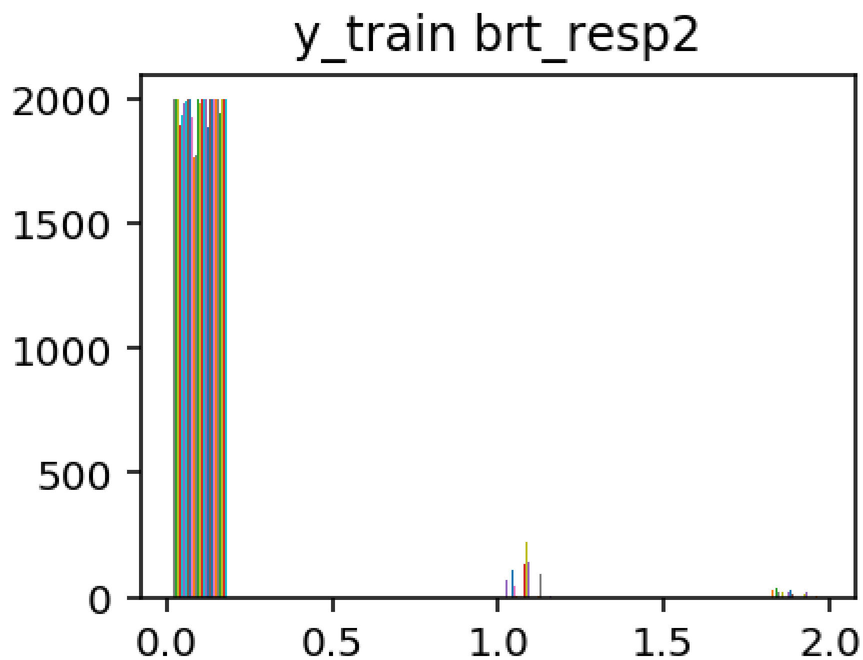


In [12]:

```python
# Plot training feature histograms
plt.rcParams['figure.figsize'] = [12.8, 6.4]
plt.rcParams['figure.dpi'] = 144

fig, axes = plt.subplots(2, 5)

for i, ax in zip(range(X_train.shape[2]), axes.flatten()):
    ax.hist(X_train[:,:,i])
    ax.set_title('X_train {}'.format(feature_list[i]))
```



In [13]:
```
# Plot training response histogram
plt.rcParams['figure.figsize'] = [3.2, 2.4]
plt.rcParams['figure.dpi'] = 144

plt.hist(y_train)
plt.title('y_train {}'.format(response_list[0]))
plt.show()
```



Random Forest classification

In [14]:
```
# Reshape arrays
X_train_nx, X_train_ny, X_train_ns = X_train.shape
X_train = X_train.reshape((X_train_nx * X_train_ny, X_train_ns))
y_train_nx, y_train_ny = y_train.shape
y_train = y_train.reshape((y_train_nx * y_train_ny))

X_test_nx, X_test_ny, X_test_ns = X_test.shape
X_test = X_test.reshape((X_test_nx * X_test_ny, X_test_ns))
y_test_nx, y_test_ny = y_test.shape
y_test = y_test.reshape((y_test_nx * y_test_ny))

print("Dimensions of X_train: {}".format(X_train.shape))
print("Dimensions of y_train: {}".format(y_train.shape))

print("Dimensions of X_test: {}".format(X_test.shape))
print("Dimensions of y_test: {}".format(y_test.shape))
```

```
Dimensions of X_train: (4000000, 10)
Dimensions of y_train: (4000000,)
Dimensions of X_test: (2000000, 10)
Dimensions of y_test: (2000000,)
```

In [15]:
```
# Unique value counts
unique_elements, counts_elements = np.unique(y_train, return_counts=True)
counts = dict(np.transpose(np.asarray((unique_elements, counts_elements))))
print("Unique value counts: {}".format(counts))
```

```
Unique value counts: {0.0: 3927024.0, 1.0: 58898.0, 2.0: 14078.0}
```

In [16]:
```
# Compute sample weights for unbalanced classes as inverse of probability
counts_sum = float(sum(counts.values()))
p_max = max(counts.values())
weights = dict((x, float(p_max)/float(y)) for x, y in counts.items())
sample_weight = [weights.get(i, i) for i in y_train]
print("Sample weights: {}".format(weights))
```

```
Sample weights: {0.0: 1.0, 1.0: 66.67499745322422, 2.0: 278.9475777809348}
```

In [17]:
```python
# Instantiate classifier
clf = RandomForestClassifier(n_estimators = 50, random_state=21, n_jobs = -2, ve

# Fit classifier to training set
clf = clf.fit(X_train, y_train, sample_weight=sample_weight)
```

```
[Parallel(n_jobs=-2)]: Using backend ThreadingBackend with 7 concurrent worker
s.

building tree 1 of 50building tree 2 of 50

building tree 3 of 50building tree 4 of 50building tree 5 of 50
building tree 6 of 50
building tree 7 of 50


building tree 8 of 50
building tree 9 of 50
building tree 10 of 50
building tree 11 of 50
building tree 12 of 50
building tree 13 of 50
building tree 14 of 50
building tree 15 of 50
building tree 16 of 50building tree 17 of 50

building tree 18 of 50
building tree 19 of 50
building tree 20 of 50
building tree 21 of 50
building tree 22 of 50
building tree 23 of 50
building tree 24 of 50
building tree 25 of 50
building tree 26 of 50
building tree 27 of 50
building tree 28 of 50
building tree 29 of 50
building tree 30 of 50
building tree 31 of 50
building tree 32 of 50
building tree 33 of 50
building tree 34 of 50
building tree 35 of 50

[Parallel(n_jobs=-2)]: Done  27 tasks      | elapsed:   22.7s

building tree 36 of 50
building tree 37 of 50
building tree 38 of 50
building tree 39 of 50
building tree 40 of 50
building tree 41 of 50
building tree 42 of 50
building tree 43 of 50
building tree 44 of 50
building tree 45 of 50
```

```
building tree 46 of 50
building tree 47 of 50building tree 48 of 50

building tree 49 of 50
building tree 50 of 50
[Parallel(n_jobs=-2)]: Done  50 out of  50 | elapsed:   39.4s finished
```

In [18]:

```
# Compute training metrics
accuracy = clf.score(X_train, y_train)

#  Predict labels of test set
train_pred = clf.predict(X_train)

# Compute MSE, confusion matrix, classification report
mse = mean_squared_error(y_train, train_pred)
conf_mat = confusion_matrix(y_train.round(), train_pred.round())
clas_rep = classification_report(y_train.round(), train_pred.round())

# Print reports
print('{:=^80}'.format('RF training report'))
print('Accuracy: %.4f' % accuracy)
print("MSE: %.4f" % mse)
print("Confusion matrix:\n{}".format(conf_mat))
print("Classification report:\n{}".format(clas_rep))
```

```
[Parallel(n_jobs=7)]: Using backend ThreadingBackend with 7 concurrent workers.
[Parallel(n_jobs=7)]: Done  27 tasks      | elapsed:    4.4s
[Parallel(n_jobs=7)]: Done  50 out of  50 | elapsed:    7.4s finished
[Parallel(n_jobs=7)]: Using backend ThreadingBackend with 7 concurrent workers.
[Parallel(n_jobs=7)]: Done  27 tasks      | elapsed:    4.4s
[Parallel(n_jobs=7)]: Done  50 out of  50 | elapsed:    7.5s finished

===============================RF training report===============================
=
Accuracy: 1.0000
MSE: 0.0000
Confusion matrix:
[[3927024       0       0]
 [      0   58898       0]
 [      0       1   14077]]
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00   3927024
         1.0       1.00      1.00      1.00     58898
         2.0       1.00      1.00      1.00     14078

    accuracy                           1.00   4000000
   macro avg       1.00      1.00      1.00   4000000
weighted avg       1.00      1.00      1.00   4000000
```

Here is how to read the above confusion matrix:

	Prediction: 0 (Unaffected)	Prediction: 1 (BA)	Prediction: 2 (PCL)

	Prediction: 0 (Unaffected)	Prediction: 1 (BA)	Prediction: 2 (PCL)
Actual: 0 (Unaffected)	**Unaffected classified as unaffected**	Unaffected classified as BA	Unaffected classified as PCL
Actual: 1 (BA)	BA classified as unaffected	**BA classified as BA**	BA classified as PCL
Actual: 2 (PCL)	PCL classified as unaffected	PCL classified as BA	**PCL classified as PCL**

BA = Burned area; PCL = Potential fire control location

In [19]:

```
# Compute testing metrics
accuracy = clf.score(X_test, y_test)

# Predict labels of test set
y_pred = clf.predict(X_test)

# Compute MSE, confusion matrix, classification report
mse = mean_squared_error(y_test, y_pred)
conf_mat = confusion_matrix(y_test.round(), y_pred.round())
clas_rep = classification_report(y_test.round(), y_pred.round())

# Print reports
print('{:=^80}'.format('RF testing report'))
print('Accuracy: %.4f' % accuracy)
print("MSE: %.4f" % mse)
print("Confusion matrix:\n{}".format(conf_mat))
print("Classification report:\n{}".format(clas_rep))
```

```
[Parallel(n_jobs=7)]: Using backend ThreadingBackend with 7 concurrent workers.
[Parallel(n_jobs=7)]: Done  27 tasks      | elapsed:    2.3s
[Parallel(n_jobs=7)]: Done  50 out of  50 | elapsed:    4.0s finished
[Parallel(n_jobs=7)]: Using backend ThreadingBackend with 7 concurrent workers.
[Parallel(n_jobs=7)]: Done  27 tasks      | elapsed:    2.1s
[Parallel(n_jobs=7)]: Done  50 out of  50 | elapsed:    3.8s finished

===============================RF testing report================================
=
Accuracy: 0.9879
MSE: 0.0121
Confusion matrix:
[[1919476       0       0]
 [      0   53137   10528]
 [      0   13585    3274]]
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00   1919476
         1.0       0.80      0.83      0.82     63665
         2.0       0.24      0.19      0.21     16859

    accuracy                           0.99   2000000
   macro avg       0.68      0.68      0.68   2000000
weighted avg       0.99      0.99      0.99   2000000
```

In [20]:
```python
# Compute predicted probabilities
y_pred_prob = clf.predict_proba(X_test)[:,1]
```

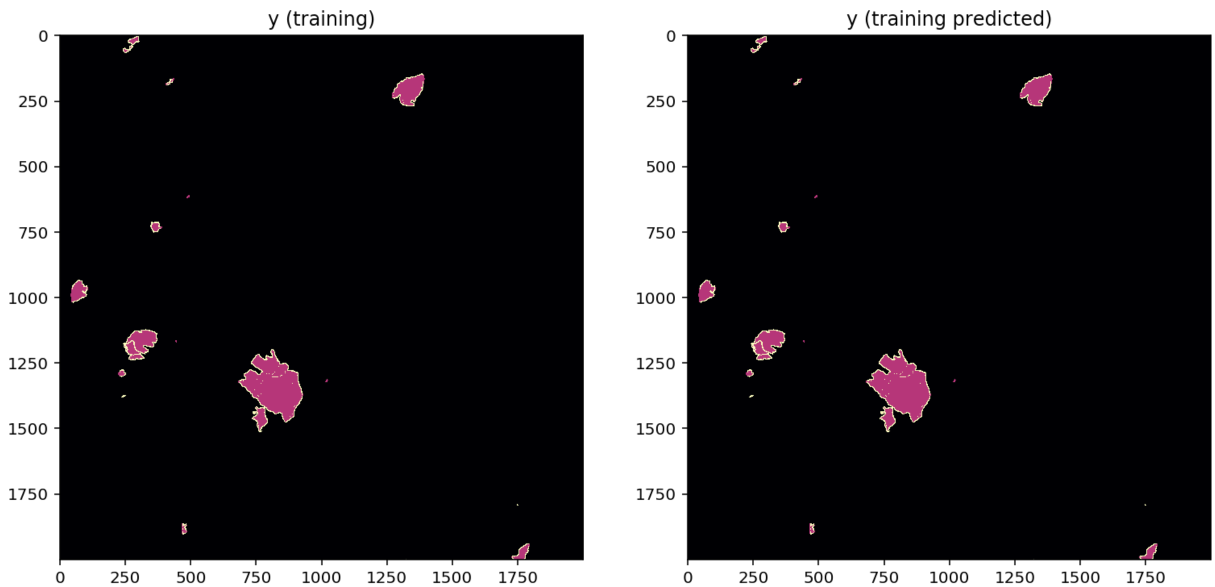
```
[Parallel(n_jobs=7)]: Using backend ThreadingBackend with 7 concurrent workers.
[Parallel(n_jobs=7)]: Done  27 tasks      | elapsed:    2.1s
[Parallel(n_jobs=7)]: Done  50 out of  50 | elapsed:    3.6s finished
```

In [21]:
```python
# Reshape arrays
X_train = X_train.reshape(X_train_nx, X_train_ny, X_train_ns)
y_train = y_train.reshape(y_train_nx, y_train_ny)
train_pred = train_pred.reshape(y_train_nx, y_train_ny)

X_test = X_test.reshape(X_test_nx, X_test_ny, X_test_ns)
y_test = y_test.reshape(y_test_nx, y_test_ny)
y_pred = y_pred.reshape(y_test_nx, y_test_ny)

print("Dimensions of X_train: {}".format(X_train.shape))
print("Dimensions of y_train: {}".format(y_train.shape))
print("Dimensions of train_pred: {}".format(train_pred.shape))

print("Dimensions of X_test: {}".format(X_test.shape))
print("Dimensions of y_test: {}".format(y_test.shape))
print("Dimensions of y_pred: {}".format(y_pred.shape))
```

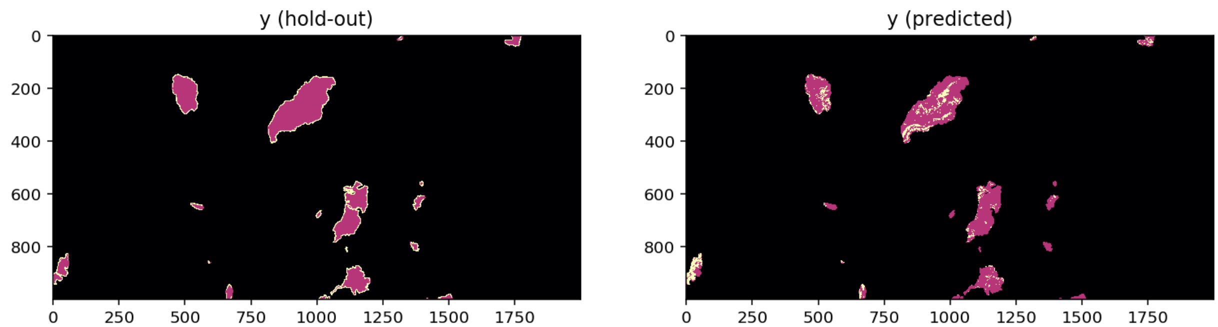
```
Dimensions of X_train: (2000, 2000, 10)
Dimensions of y_train: (2000, 2000)
Dimensions of train_pred: (2000, 2000)
Dimensions of X_test: (1000, 2000, 10)
Dimensions of y_test: (1000, 2000)
Dimensions of y_pred: (1000, 2000)
```

In [22]:

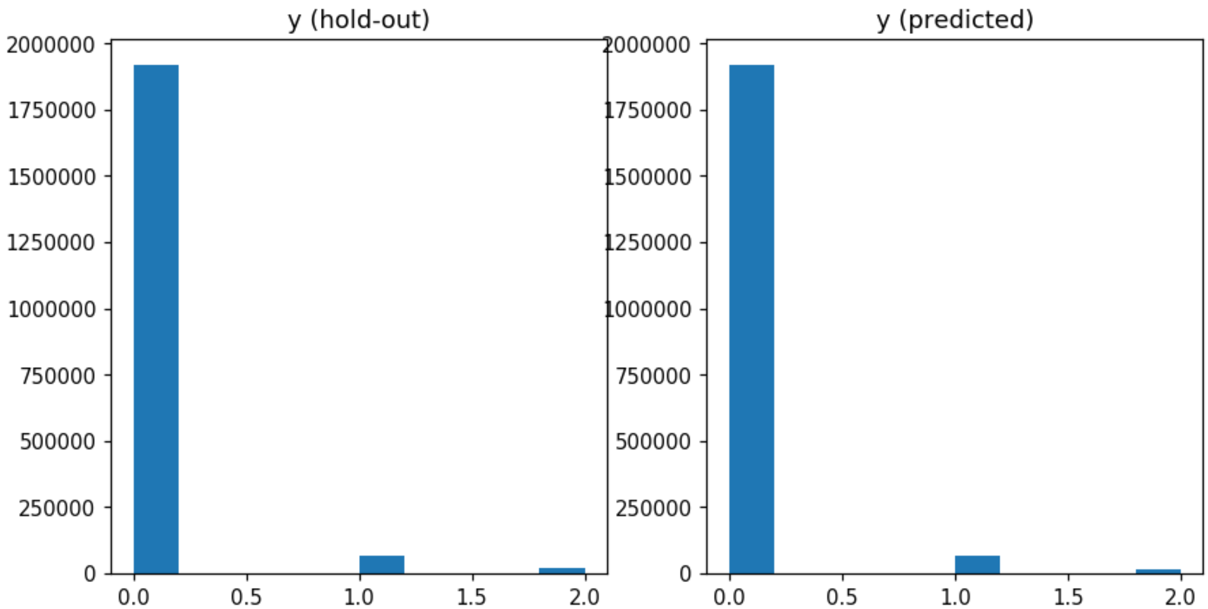
```
# Plot training data and prediction maps
plt.rcParams['figure.figsize'] = [12.8, 9.6]
plt.rcParams['figure.dpi'] = 144

plt.subplot(121)
plt.imshow(y_train, cmap=plt.cm.get_cmap('magma'))
plt.title('y (training)')

plt.subplot(122)
plt.imshow(train_pred, cmap=plt.cm.get_cmap('magma'))
plt.title('y (training predicted)')

plt.show()
```



In [23]:
```
# Plot test data and prediction maps
plt.rcParams['figure.figsize'] = [12.8, 9.6]
plt.rcParams['figure.dpi'] = 144

plt.subplot(121)
plt.imshow(y_test, cmap=plt.cm.get_cmap('magma'))
plt.title('y (hold-out)')

plt.subplot(122)
plt.imshow(y_pred, cmap=plt.cm.get_cmap('magma'))
plt.title('y (predicted)')

plt.show()
```

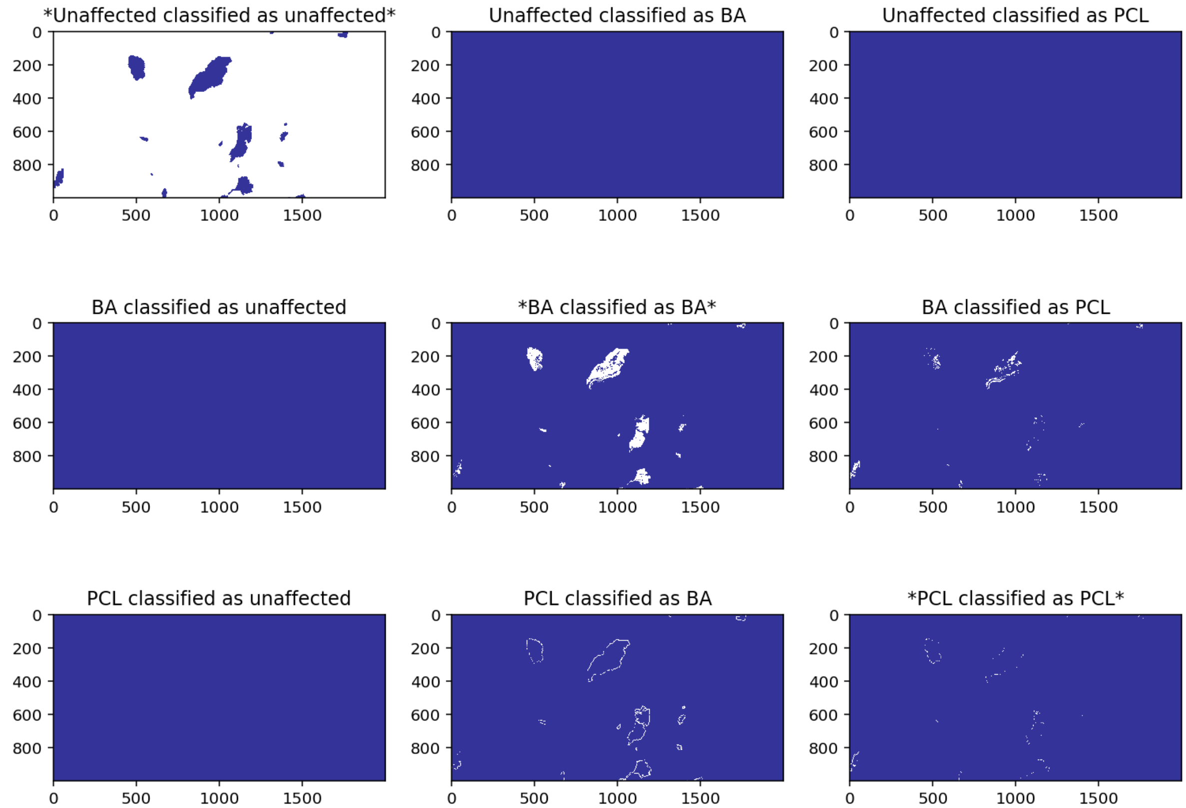


In [24]:
```
# Plot histograms of test data and prediction
plt.rcParams['figure.figsize'] = [9.6, 4.8]
plt.rcParams['figure.dpi'] = 108

plt.subplot(121)
plt.hist(y_test.flatten())
plt.title('y (hold-out)')

plt.subplot(122)
plt.hist(y_pred.flatten())
plt.title('y (predicted)')

plt.show()
```



In [25]:
```
# Create predicted condition arrays
y_pred_00 = (y_test.round() == 0) & (y_pred.round() == 0)
y_pred_01 = (y_test.round() == 0) & (y_pred.round() == 1)
y_pred_02 = (y_test.round() == 0) & (y_pred.round() == 2)

y_pred_10 = (y_test.round() == 1) & (y_pred.round() == 0)
y_pred_11 = (y_test.round() == 1) & (y_pred.round() == 1)
y_pred_12 = (y_test.round() == 1) & (y_pred.round() == 2)

y_pred_20 = (y_test.round() == 2) & (y_pred.round() == 0)
y_pred_21 = (y_test.round() == 2) & (y_pred.round() == 1)
y_pred_22 = (y_test.round() == 2) & (y_pred.round() == 2)
```

In [26]:
```
# Plot predicted condition maps
plt.rcParams['figure.figsize'] = [12.8, 9.6]
plt.rcParams['figure.dpi'] = 144

plt.subplot(331)
plt.imshow(y_pred_00, cmap=plt.cm.terrain)
plt.title('*Unaffected classified as unaffected*')

plt.subplot(332)
plt.imshow(y_pred_01, cmap=plt.cm.terrain)
plt.title('Unaffected classified as BA')

plt.subplot(333)
plt.imshow(y_pred_02, cmap=plt.cm.terrain)
plt.title('Unaffected classified as PCL')

plt.subplot(334)
plt.imshow(y_pred_10, cmap=plt.cm.terrain)
plt.title('BA classified as unaffected')

plt.subplot(335)
plt.imshow(y_pred_11, cmap=plt.cm.terrain)
plt.title('*BA classified as BA*')

plt.subplot(336)
plt.imshow(y_pred_12, cmap=plt.cm.terrain)
plt.title('BA classified as PCL')

plt.subplot(337)
plt.imshow(y_pred_20, cmap=plt.cm.terrain)
plt.title('PCL classified as unaffected')

plt.subplot(338)
plt.imshow(y_pred_21, cmap=plt.cm.terrain)
plt.title('PCL classified as BA')

plt.subplot(339)
plt.imshow(y_pred_22, cmap=plt.cm.terrain)
plt.title('*PCL classified as PCL*')

plt.show()
```



Feature importance

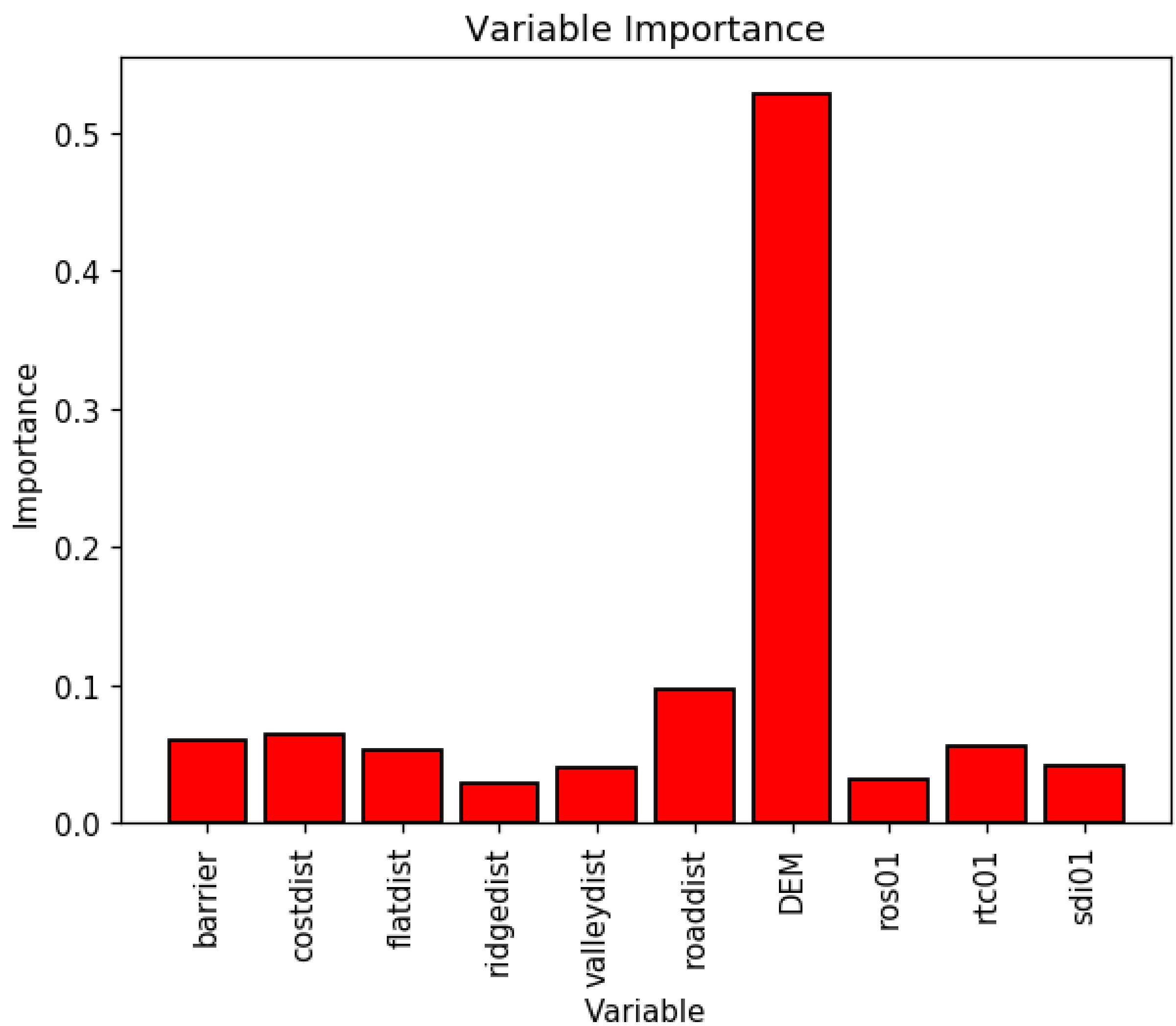
In [27]:

```
# Get feature importances
importances = list(clf.feature_importances_)
feature_importances = [(feature, round(importance, 4)) for feature, importance in

# Sort feature importance in descending order
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse =

# Print feature importance
_ = [print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_im
```

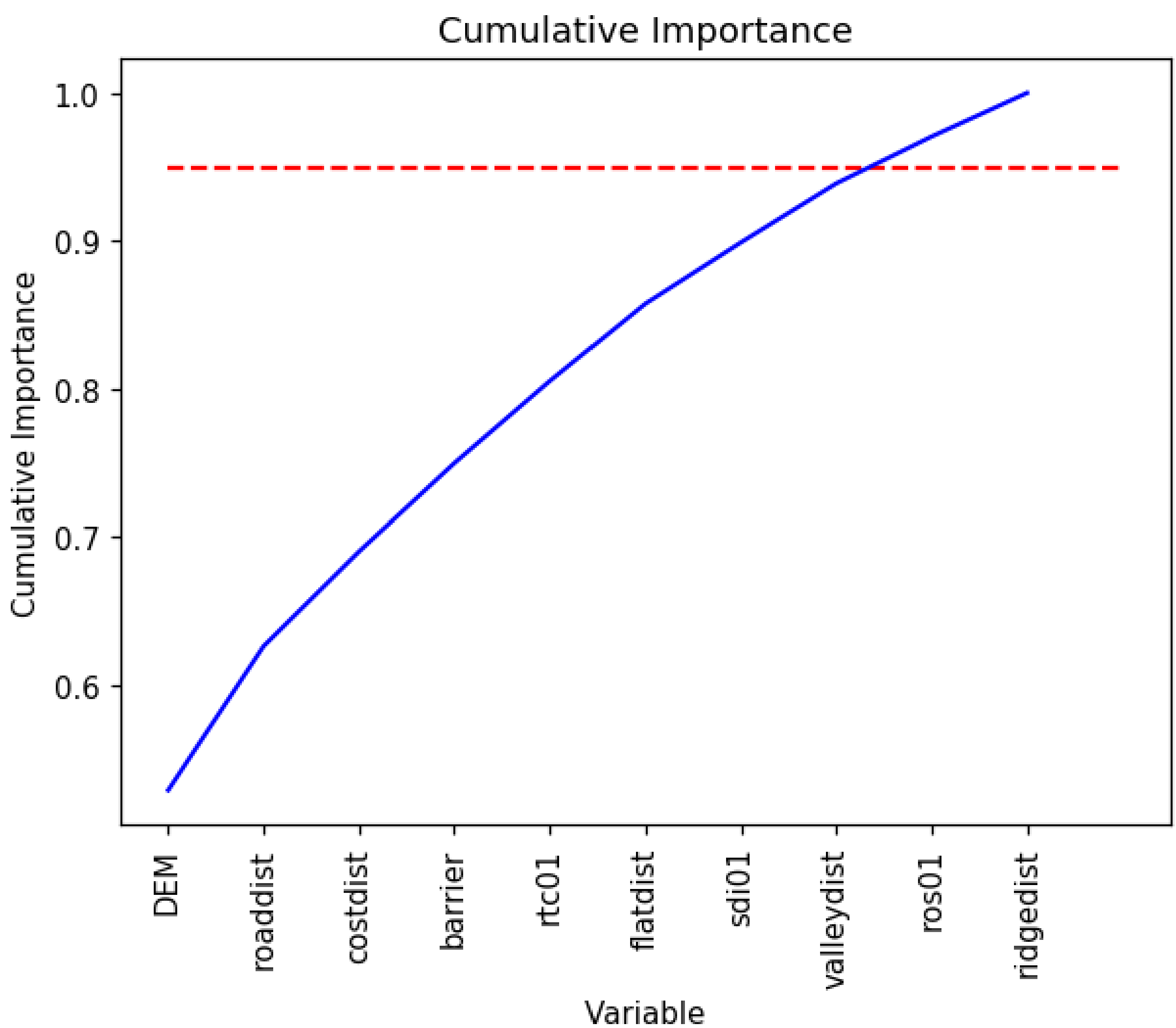
```
Variable: DEM                  Importance: 0.5288
Variable: roaddist             Importance: 0.0974
Variable: costdist             Importance: 0.0639
Variable: barrier              Importance: 0.0596
Variable: rtc01                Importance: 0.0555
Variable: flatdist             Importance: 0.0524
Variable: sdi01                Importance: 0.0414
Variable: valleydist           Importance: 0.0399
Variable: ros01                Importance: 0.0317
Variable: ridgedist            Importance: 0.0295
```

In [28]:

```
# Bar plot of relative importance
plt.rcParams['figure.figsize'] = [6.4, 4.8]
plt.rcParams['figure.dpi'] = 108

X_values = list(range(len(importances)))
plt.bar(X_values, importances, orientation = 'vertical', color = 'r', edgecolor =
plt.xticks(X_values, feature_list, rotation = 'vertical')
plt.ylabel('Importance')
plt.xlabel('Variable')
plt.title('Variable Importance')
plt.show()
```



In [29]:
```
# List of features sorted by decreasing importance
sorted_importances = [importance[1] for importance in feature_importances]
sorted_features = [importance[0] for importance in feature_importances]

# Cumulative importance
cumulative_importances = np.cumsum(sorted_importances)

# Create line plot
plt.plot(X_values, cumulative_importances, 'b-')
plt.hlines(y = 0.95, xmin=0, xmax=len(sorted_importances), color = 'r', linestyl
plt.xticks(X_values, sorted_features, rotation = 'vertical')
plt.xlabel('Variable')
plt.ylabel('Cumulative Importance')
plt.title('Cumulative Importance')
plt.show()
```



In [30]:
```
# Number of features explaining 95% cum. importance
n_import = np.where(cumulative_importances > 0.95)[0][0] + 1
print('Number of features required (95% importance):', n_import)
```

```
Number of features required (95% importance): 9
```

Classifier optimization

In [31]:
```
# Extract the names of most important features
important_feature_names = [feature[0] for feature in feature_importances[0:(n_im

# Column indices of most important features
important_indices = [feature_list.index(feature) for feature in important_featur

# Create training and testing sets with only important features
X_train_imp = X_train[:, :, important_indices]
X_test_imp = X_test[:, :, important_indices]
```

In [32]:
```
# Reshape arrays
X_train_imp_nx, X_train_imp_ny, X_train_imp_ns = X_train_imp.shape
X_train_imp = X_train_imp.reshape((X_train_imp_nx * X_train_imp_ny, X_train_imp_
y_train_nx, y_train_ny = y_train.shape
y_train = y_train.reshape((y_train_nx * y_train_ny))

X_test_imp_nx, X_test_imp_ny, X_test_imp_ns = X_test_imp.shape
X_test_imp = X_test_imp.reshape((X_test_imp_nx * X_test_imp_ny, X_test_imp_ns))
y_test_nx, y_test_ny = y_test.shape
y_test = y_test.reshape((y_test_nx * y_test_ny))

print("Dimensions of X_train: {}".format(X_train_imp.shape))
print("Dimensions of y_train: {}".format(y_train.shape))
print("Dimensions of X_test: {}".format(X_test_imp.shape))
print("Dimensions of y_test: {}".format(y_test.shape))
```

```
Dimensions of X_train: (4000000, 8)
Dimensions of y_train: (4000000,)
Dimensions of X_test: (2000000, 8)
Dimensions of y_test: (2000000,)
```

In [33]:
```python
# Fit classifier to training set
clf = clf.fit(X_train_imp, y_train, sample_weight=sample_weight)
```

```
[Parallel(n_jobs=-2)]: Using backend ThreadingBackend with 7 concurrent worker
s.

building tree 1 of 50building tree 2 of 50
building tree 3 of 50building tree 4 of 50

building tree 5 of 50building tree 6 of 50

building tree 7 of 50

building tree 8 of 50
building tree 9 of 50
building tree 10 of 50
building tree 11 of 50
building tree 12 of 50
building tree 13 of 50
building tree 14 of 50
building tree 15 of 50
building tree 16 of 50
building tree 17 of 50
building tree 18 of 50
building tree 19 of 50
building tree 20 of 50
building tree 21 of 50
building tree 22 of 50
building tree 23 of 50
building tree 24 of 50
building tree 25 of 50
building tree 26 of 50
building tree 27 of 50
building tree 28 of 50
building tree 29 of 50
building tree 30 of 50
building tree 31 of 50
building tree 32 of 50
building tree 33 of 50
building tree 34 of 50

[Parallel(n_jobs=-2)]: Done  27 tasks      | elapsed:   15.0s

building tree 35 of 50
building tree 36 of 50
building tree 37 of 50
building tree 38 of 50building tree 39 of 50

building tree 40 of 50
building tree 41 of 50building tree 42 of 50

building tree 43 of 50
building tree 44 of 50
building tree 45 of 50
building tree 46 of 50
building tree 47 of 50building tree 48 of 50
```

```
building tree 49 of 50
building tree 50 of 50

[Parallel(n_jobs=-2)]: Done  50 out of  50 | elapsed:   27.2s finished
```

In [34]:

```python
# Compute training metrics
accuracy = clf.score(X_train_imp, y_train)

#  Predict labels of test set
train_pred = clf.predict(X_train_imp)

# Compute MSE, confusion matrix, classification report
mse = mean_squared_error(y_train, train_pred)
conf_mat = confusion_matrix(y_train.round(), train_pred.round())
clas_rep = classification_report(y_train.round(), train_pred.round())

# Print reports
print('{:=^80}'.format('RF training report'))
print('Accuracy: %.4f' % accuracy)
print("MSE: %.4f" % mse)
print("Confusion matrix:\n{}".format(conf_mat))
print("Classification report:\n{}".format(clas_rep))
```

```
[Parallel(n_jobs=7)]: Using backend ThreadingBackend with 7 concurrent workers.
[Parallel(n_jobs=7)]: Done  27 tasks      | elapsed:    3.9s
[Parallel(n_jobs=7)]: Done  50 out of  50 | elapsed:    6.6s finished
[Parallel(n_jobs=7)]: Using backend ThreadingBackend with 7 concurrent workers.
[Parallel(n_jobs=7)]: Done  27 tasks      | elapsed:    3.9s
[Parallel(n_jobs=7)]: Done  50 out of  50 | elapsed:    6.5s finished

===============================RF training report===============================
=
Accuracy: 1.0000
MSE: 0.0000
Confusion matrix:
[[3927024       0       0]
 [      0   58897       1]
 [      0       0   14078]]
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00   3927024
         1.0       1.00      1.00      1.00     58898
         2.0       1.00      1.00      1.00     14078

    accuracy                           1.00   4000000
   macro avg       1.00      1.00      1.00   4000000
weighted avg       1.00      1.00      1.00   4000000
```

In [35]:
```
# Compute testing metrics
accuracy = clf.score(X_test_imp, y_test)

# Predict labels of test set
y_pred = clf.predict(X_test_imp)

# Compute MSE, confusion matrix, classification report
mse = mean_squared_error(y_test, y_pred)
conf_mat = confusion_matrix(y_test.round(), y_pred.round())
clas_rep = classification_report(y_test.round(), y_pred.round())

# Print reports
print('{:=^80}'.format('RF testing report'))
print('Accuracy: %.4f' % accuracy)
print("MSE: %.4f" % mse)
print("Confusion matrix:\n{}".format(conf_mat))
print("Classification report:\n{}".format(clas_rep))
```

```
[Parallel(n_jobs=7)]: Using backend ThreadingBackend with 7 concurrent workers.
[Parallel(n_jobs=7)]: Done  27 tasks      | elapsed:    1.8s
[Parallel(n_jobs=7)]: Done  50 out of  50 | elapsed:    3.4s finished
[Parallel(n_jobs=7)]: Using backend ThreadingBackend with 7 concurrent workers.
[Parallel(n_jobs=7)]: Done  27 tasks      | elapsed:    1.6s
[Parallel(n_jobs=7)]: Done  50 out of  50 | elapsed:    3.1s finished

===============================RF testing report===============================
=
Accuracy: 0.9872
MSE: 0.0128
Confusion matrix:
[[1919476       0       0]
 [      0   51460   12205]
 [      0   13373    3486]]
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00   1919476
         1.0       0.79      0.81      0.80     63665
         2.0       0.22      0.21      0.21     16859

    accuracy                           0.99   2000000
   macro avg       0.67      0.67      0.67   2000000
weighted avg       0.99      0.99      0.99   2000000
```

In [36]:
```
# Compute predicted probabilities
y_pred_prob = clf.predict_proba(X_test_imp)[:,1]
```

```
[Parallel(n_jobs=7)]: Using backend ThreadingBackend with 7 concurrent workers.
[Parallel(n_jobs=7)]: Done  27 tasks      | elapsed:    1.8s
[Parallel(n_jobs=7)]: Done  50 out of  50 | elapsed:    3.0s finished
```