Archeological Site Prediction

May 8, 2018

1 Archeological Site Prediction Project

by **Catherine Yeager** and **William Cheng**
198:439 Intro to Data Science, Spring 2018
Professor Gerard de Melo

1.1 Objective

The purpose of this project is to predict the archeological site locations within the neolithic era. We want to extend the current method of discovering archeological sites inside a small area to an entire country.

So, we've decided to **predict neolithic archeological sites within the entire country of Ireland!**

1.2 Data

We collected the archeological site locations in Europe from the EUROVOL Dataset, and we extracted the site locations that were located in Ireland. The elevation feature was obtained from open source NASA SRTM databases, and the slope and aspect were derived from this. The distances were calculated from using shape files of lakes and rivers obtained from the open source databsse naturalearthdata. The climate data that was collected was also from an open source database

We filtered through, mined, and calculated data on the following attributes - Elevation - Slope - Aspect - Distance to lakes - Distance to rivers - Distance to coastlines/borders - Annual Mean Temperature - Mean Diurnal Range (Mean of monthly (max temp - min temp)) - Isothermality (BIO2/BIO7) (* 100) - Temperature Seasonality (standard deviation *100) - Max Temperature of Warmest Month - Min Temperature of Coldest Month - Temperature Annual Range (BIO5-BIO6) - Mean Temperature of Wettest Quarter - Mean Temperature of Driest Quarter - Mean Temperature of Warmest Quarter - Mean Temperature of Coldest Quarter - Annual Precipitation - Precipitation of Wettest Month - Precipitation of Driest Month - Precipitation Seasonality (Coefficient of Variation) - Precipitation of Wettest Quarter - Precipitation of Driest Quarter - Precipitation of Warmest Quarter - Precipitation of Coldest Quarter

2 Data Preprocessing

Import the data we're using to train from `common_sites_V3.csv` and the data we want to predict from `ireland_sites_V3.csv`

However, the data we have in `common_sites_V3.csv` only has *positive values (locations of archeological sites)*, so we have to sample random locations in Ireland and say that those locations do not have archeological sites

The command below imports the function that automatically takes samples of nonsite coordinates, since we only have coordinates where archeological sites have been found

```
from randomSiteSampling import randomSamplingFromDF
```

The training data we use are from both **Ireland** and **Great Britain** since the two countries are so close in proximity, and we are only trying to discover archeological sites in **Ireland**

```
In [147]: import time
          import pandas as pd
          import sklearn as ml
          import matplotlib
          %matplotlib inline

          import pprint
          pp = pprint.PrettyPrinter(indent=4).pprint

          from randomSiteSampling import randomSamplingFromDF

          raw_site_file = 'common_sites_V3.csv'
          raw_pred_file = 'ireland_sites_V3.csv'

          #Import data
          raw_site_df = pd.read_csv(raw_site_file,
                                    low_memory=False,
                                    error_bad_lines=False,
                                    encoding='ISO-8859-1')
          raw_pred_df = pd.read_csv(raw_pred_file,
                                    low_memory=False,
                                    error_bad_lines=False,
                                    encoding='ISO-8859-1')
```

Since there only exists information on sites that have been discovered, there does not seem to be an extensive database on the coordinates where there is certainly no sites at all.

Thus, we sample randomly from coordinates in Ireland, and we say there are no sites at these points.

`nonsite_df` and `pred_df` hold the randomly sampled coordiantes where we say there are no sites at these coordinates and ireland coordinates without those same randomly sampled coordinates, respectively

```
In [119]: nonsite_df, pred_df = randomSamplingFromDF(raw_site_df, raw_pred_df)
```

The `predictions` variable is used to hold outputs from different models because it will still contain the latitude and longitude values

The `site_df` variable will be the one that will be mutated to split between training and testing data

```
In [120]: # copy pred_df to predictions
          predictions = pred_df.copy()

          # copy raw_site_df to site_df
          site_df = raw_site_df.copy()
```

Making sure all dataframes have the same columns We do this by finding all the columns which are not shared between all dataframes. We then keep only the columns common to all the dataframes

```
In [121]: site_attr      = set(site_df)
          nonsite_attr   = set(nonsite_df)
          pred_attr      = set(pred_df)

          common_attr    = site_attr & nonsite_attr & pred_attr
          uncommon_attr = (site_attr | nonsite_attr | pred_attr) - common_attr

          # create list of columns to drop
          drop_site      = (site_attr    & uncommon_attr) | {'latitude', 'longitude'}
          drop_nonsite   = (nonsite_attr & uncommon_attr) | {'latitude', 'longitude'}
          drop_pred      = (pred_attr    & uncommon_attr) | {'latitude', 'longitude'}

          # drop columns
          site_df        = site_df.drop(columns=drop_site)
          nonsite_df     = nonsite_df.drop(columns=drop_nonsite)
          pred_df        = pred_df.drop(columns=drop_pred)
```

We have some NA data in our climate columns. However, the set of coordinates within Ireland that we are working with have neighbors that are about `300m away`, so we **interpolate** the climate data from the closest neighbors using the two methods in succession to fill the data forward and then backward

```
pd.DataFrame.fillna(method='ffill')
pd.DataFrame.fillna(method='bfill')
```

```
In [122]: site_df        = site_df.fillna    (method='ffill').fillna(method='bfill')
          nonsite_df     = nonsite_df.fillna(method='ffill').fillna(method='bfill')
          pred_df        = pred_df.fillna    (method='ffill').fillna(method='bfill')
```

Here we label our data appropriately for training.
We also do a full outer join on our `site_df` and our randomly sampled `nonsite_df`

```
pd.concat([site_df, nonsite_df])
```

```
In [123]: site_df['site'] = 1
          nonsite_df['site'] = 0
          data = pd.concat([site_df, nonsite_df])

          target = data['site']
          data = data.drop(columns=['site'])
```

3 Machine Learning

We split the data into the corresponding **training sets** and **testing sets**
The ratio of the training set to the testing test is determined by the variable`train_size`

```
In [160]: from sklearn.model_selection import train_test_split

          train_size=0.7

          train_data, test_data, train_target, test_target = train_test_split(data,
                                                                          target,
                                                                          train_size=train_s
```

We use the following **Machine Learning models** to train the data with:
1. SVC Model (Linear kernel)
2. K-Neighbors Classifier
3. Decision Forest Classifier
4. Random Forest Classifier
5. Gradient Boosting Classifier
6. Ada Boosting Classifier

The following function **plotPredictedFromDF** is used to automatically visualize the resulting data from each model

```
In [125]: from plotPredicted import plotPredictedFromDF
```

3.0.1 1. Linear SVC Model

`1a. Training the Linear SVC Model`

```
In [126]: from sklearn import svm

          SVCModel = svm.SVC(kernel='linear', C=0.5)

          start_time = time.time()
          SVCModel.fit(train_data, train_target)
          print "Time to Train: %f seconds" % (time.time() - start_time)
```

```
Time to Train: 11.833799 seconds
```

`1b. Testing the Linear SVC Model`

```
In [127]: from sklearn import metrics

          model = SVCModel
          """
          Fit the training data
          and observe the metrics
          """
```

```
        preds = model.predict(train_data)
        targs = train_target
        args = (targs, preds)
        print "Training Data"
        print "accuracy  : %.03f%%" % (metrics.accuracy_score(*args)*100)
        print "precision : %.03f%%" % (metrics.precision_score(*args)*100)
        print "recall    : %.03f%%" % (metrics.recall_score(*args)*100)
        print "f1        : %.03f%%" % (metrics.f1_score(*args)*100)

        """
        Fit the test data
        and observe the metrics
        """
        preds = model.predict(test_data)
        targs = test_target
        args = (targs, preds)
        print "\n"
        print "Test Data"
        print "accuracy  : %.03f%%" % (metrics.accuracy_score(*args)*100)
        print "precision : %.03f%%" % (metrics.precision_score(*args)*100)
        print "recall    : %.03f%%" % (metrics.recall_score(*args)*100)
        print "f1        : %.03f%%" % (metrics.f1_score(*args)*100)
```

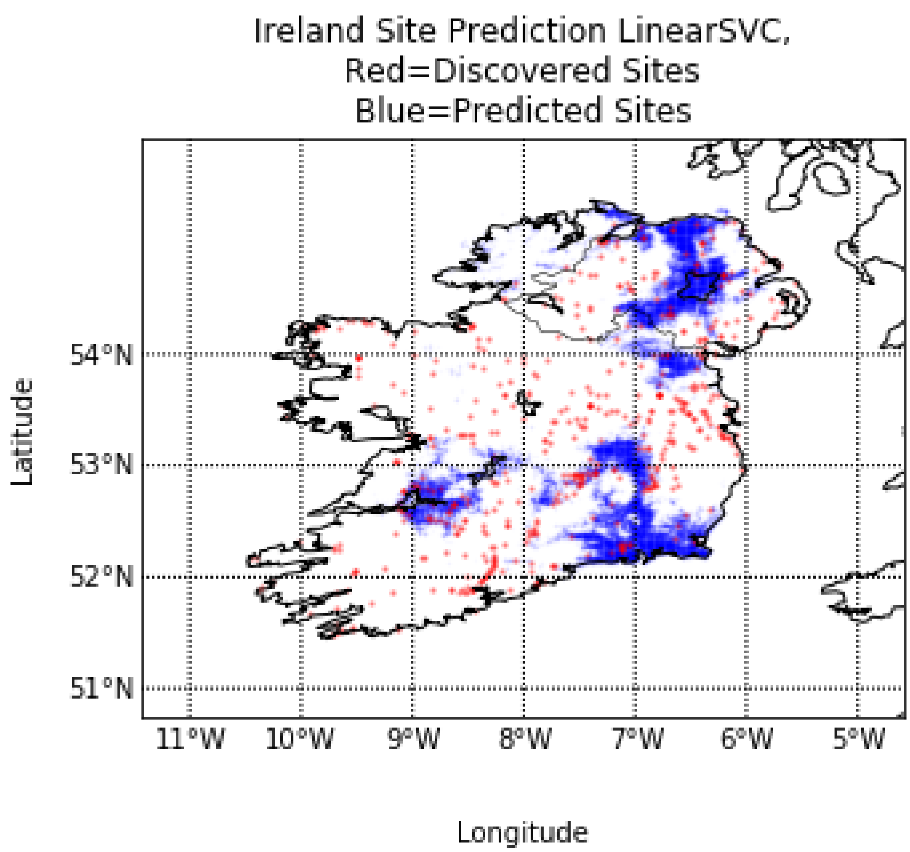
```
Training Data
accuracy  : 85.701%
precision : 89.720%
recall    : 80.823%
f1        : 85.039%


Test Data
accuracy  : 83.443%
precision : 86.199%
recall    : 79.111%
f1        : 82.503%
```

1c. Linear SVC Prediction

```
In [161]: model_name = 'LinearSVC'
          predictions[model_name] = SVCModel.predict(pred_df)

          site_exists = predictions[predictions[model_name] == 1]
          site_nexists = predictions[predictions[model_name] != 1]
          num_site_exists = site_exists[model_name].count()
          num_site_nexists = site_nexists[model_name].count()

          print 'Sites exists         : %d ' % (num_site_exists)
```

```
print 'Sites does not exists: %d ' % (num_site_nexists)
print 'Percentage of sites predicted having an archeological site: %.3f%s' % \
(100.0*float(num_site_exists)/float(num_site_exists+num_site_nexists), '%')

# plot the map
plotPredictedFromDF(raw_site_df,
                    predictions,
                    model_name=model_name,
                    country='Ireland',
                    resolution='i',
                    alpha_predicted=.01,
                    alpha_common=0.3)
```

```
Sites exists          : 77660
Sites does not exists: 581277
Percentage of sites predicted having an archeological site: 11.786%
```



3.0.2 2. K-Neighbors Classifier

2a. Training the K-Neighbors Classifier Model

```
In [129]: from sklearn import neighbors

          n_neighbors=20
          KNeighborsModel = neighbors.KNeighborsClassifier(n_neighbors=n_neighbors)

          start_time = time.time()
          KNeighborsModel.fit(train_data, train_target)
          print "Time to Train: %f seconds" % (time.time() - start_time)
```

```
Time to Train: 0.003130 seconds
```

2b. Testing the K-Neighbors Classifier Model

```
In [130]: from sklearn import metrics

          model = KNeighborsModel
          """
          Fit the training data
          and observe the metrics
          """
          preds = model.predict(train_data)
          targs = train_target
          args = (targs, preds)
          print "Training Data"
          print "accuracy  : %.03f%s" % (metrics.accuracy_score(*args)*100, "%")
          print "precision : %.03f%s" % (metrics.precision_score(*args)*100, "%")
          print "recall    : %.03f%s" % (metrics.recall_score(*args)*100, "%")
          print "f1        : %.03f%s" % (metrics.f1_score(*args)*100, "%")

          """
          Fit the test data
          and observe the metrics
          """
          preds = model.predict(test_data)
          targs = test_target
          args = (targs, preds)
          print "\n"
          print "Test Data"
          print "accuracy  : %.03f%s" % (metrics.accuracy_score(*args)*100, "%")
          print "precision : %.03f%s" % (metrics.precision_score(*args)*100, "%")
          print "recall    : %.03f%s" % (metrics.recall_score(*args)*100, "%")
          print "f1        : %.03f%s" % (metrics.f1_score(*args)*100, "%")
```

```
Training Data
accuracy  : 85.325%
precision : 97.019%
recall    : 73.059%
f1        : 83.351%
```

```
Test Data
accuracy  : 83.882%
precision : 94.955%
recall    : 71.111%
f1        : 81.321%
```

2c. K-Neighbors Prediction

```
In [ ]: model_name = 'KNeighborsModel'
        predictions[model_name] = KNeighborsModel.predict(pred_df)

        site_exists      = predictions[predictions[model_name] == 1]
        site_nexists     = predictions[predictions[model_name] != 1]
        num_site_exists  = site_exists[model_name].count()
        num_site_nexists = site_nexists[model_name].count()

        print 'Sites exists         : %d ' % (num_site_exists)
        print 'Sites does not exists: %d ' % (num_site_nexists)
        print 'Percentage of sites predicted having an archeological site: %.3f%s' % \
        (100.0*float(num_site_exists)/float(num_site_exists+num_site_nexists), '%')

        # plot the map
        plotPredictedFromDF(raw_site_df,
                            predictions,
                            model_name=model_name,
                            country='Ireland',
                            resolution='i',
                            alpha_predicted=.02,
                            alpha_common=0.3)
```

```
Sites exists         : 21612
Sites does not exists: 637325
Percentage of sites predicted having an archeological site: 3.280%
```

3.0.3 3. Decision Tree Classification

3a. Training the Decision Tree Classifier Model

```
In [132]: from sklearn import tree

          max_depth=8
          DecisionTree = tree.DecisionTreeClassifier(max_depth=max_depth)

          start_time = time.time()
          DecisionTree.fit(train_data, train_target)
          print "Time to Train: %f seconds" % (time.time() - start_time)
```

```
Time to Train: 0.014474 seconds
```

3b. Testing the Decision Tree Classifier Model

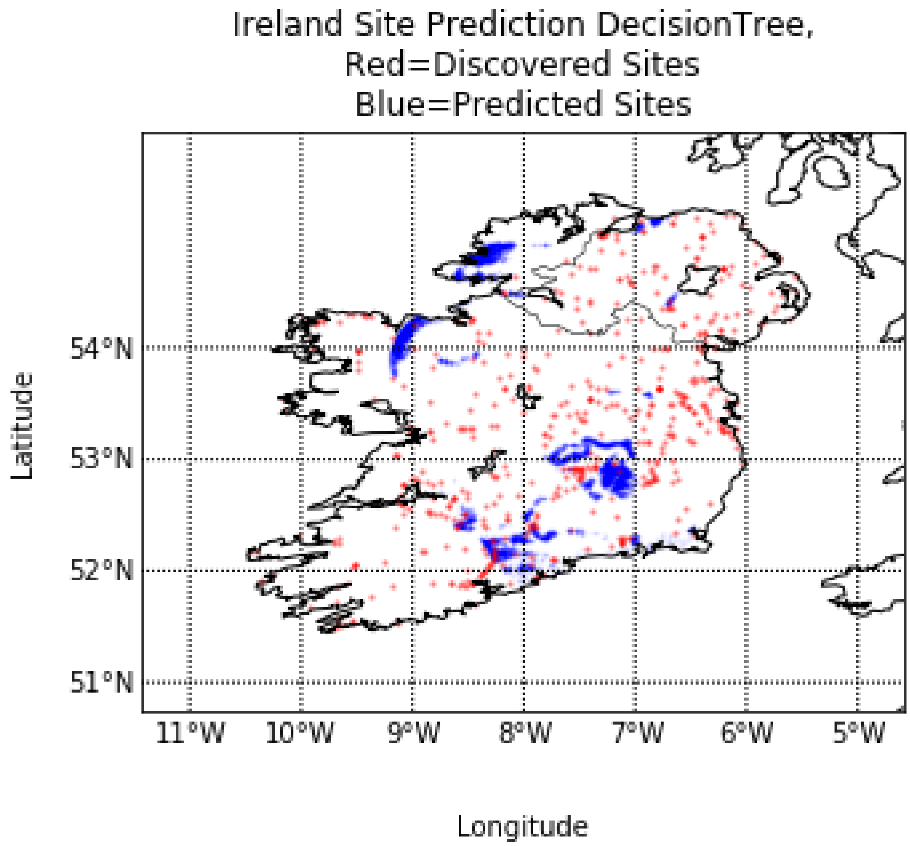
```
In [133]: from sklearn import metrics

          model = DecisionTree
          """
          Fit the training data
          and observe the metrics
          """
          preds = model.predict(train_data)
          targs = train_target
          args = (targs, preds)
          print "Training Data"
          print "accuracy  : %.03f%s" % (metrics.accuracy_score(*args)*100, "%")
          print "precision : %.03f%s" % (metrics.precision_score(*args)*100, "%")
          print "recall    : %.03f%s" % (metrics.recall_score(*args)*100, "%")
          print "f1        : %.03f%s" % (metrics.f1_score(*args)*100, "%")

          """
          Fit the test data
          and observe the metrics
          """
          preds = model.predict(test_data)
          targs = test_target
          args = (targs, preds)
          print "\n"
          print "Test Data"
          print "accuracy  : %.03f%s" % (metrics.accuracy_score(*args)*100, "%")
          print "precision : %.03f%s" % (metrics.precision_score(*args)*100, "%")
          print "recall    : %.03f%s" % (metrics.recall_score(*args)*100, "%")
          print "f1        : %.03f%s" % (metrics.f1_score(*args)*100, "%")
```

```
Training Data
accuracy  : 94.450%
precision : 98.869%
recall    : 89.991%
f1        : 94.221%


Test Data
accuracy  : 91.667%
precision : 96.059%
recall    : 86.667%
f1        : 91.121%
```

3c. Decision Tree Prediction

```
In [155]: model_name = 'DecisionTree'
          predictions[model_name] = DecisionTree.predict(pred_df)

          site_exists      = predictions[predictions[model_name] == 1]
          site_nexists     = predictions[predictions[model_name] != 1]
          num_site_exists  = site_exists[model_name].count()
          num_site_nexists = site_nexists[model_name].count()

          print 'Sites exists         : %d ' % (num_site_exists)
          print 'Sites does not exists: %d ' % (num_site_nexists)
          print 'Percentage of sites predicted having an archeological site: %.3f%s' % \
          (100.0*float(num_site_exists)/float(num_site_exists+num_site_nexists), '%')

          # plot the map
          plotPredictedFromDF(raw_site_df,
                              predictions,
                              model_name=model_name,
                              country='Ireland',
                              resolution='i',
                              alpha_predicted=.02,
                              alpha_common=0.3)
```

```
Sites exists         : 18601
Sites does not exists: 640336
Percentage of sites predicted having an archeological site: 2.823%
```



3.0.4 4. Random Forest Classification

4a. Training the Random Forest Classifier Model

```
In [135]: from sklearn import ensemble

          n_estimators=10
          max_depth=9
          RandomForest = ensemble.RandomForestClassifier(n_estimators=n_estimators,
                                                         max_depth=max_depth)

          start_time = time.time()
          RandomForest.fit(train_data, train_target)
          print "Time to Train: %f seconds" % (time.time() - start_time)
```

```
Time to Train: 0.059825 seconds
```

4b. Testing the Random Forest Classifier Model

```
In [136]: from sklearn import metrics

          model = RandomForest
          """
          Fit the training data
          and observe the metrics
          """
          preds = model.predict(train_data)
          targs = train_target
          args = (targs, preds)
          print "Training Data"
          print "accuracy  : %.03f%s" % (metrics.accuracy_score(*args)*100, "%")
          print "precision : %.03f%s" % (metrics.precision_score(*args)*100, "%")
          print "recall    : %.03f%s" % (metrics.recall_score(*args)*100, "%")
          print "f1        : %.03f%s" % (metrics.f1_score(*args)*100, "%")

          """
          Fit the test data
          and observe the metrics
          """
          preds = model.predict(test_data)
          targs = test_target
          args = (targs, preds)
          print "\n"
          print "Test Data"
          print "accuracy  : %.03f%s" % (metrics.accuracy_score(*args)*100, "%")
          print "precision : %.03f%s" % (metrics.precision_score(*args)*100, "%")
          print "recall    : %.03f%s" % (metrics.recall_score(*args)*100, "%")
          print "f1        : %.03f%s" % (metrics.f1_score(*args)*100, "%")
```

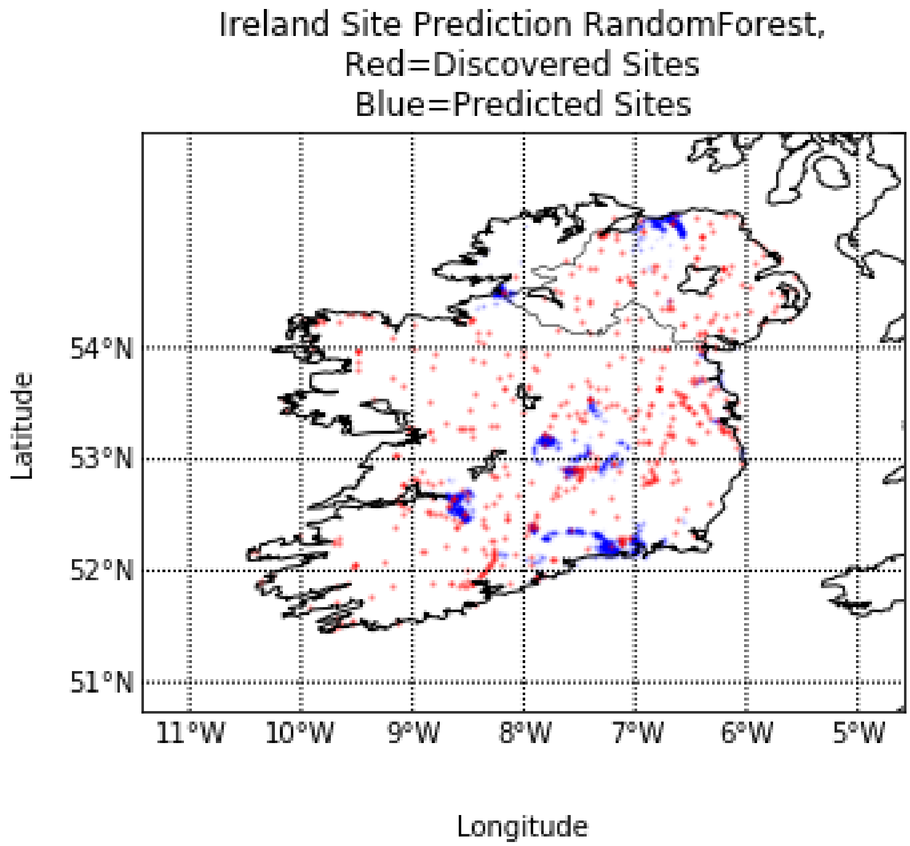
```
Training Data
accuracy  : 94.685%
precision : 99.792%
recall    : 89.616%
f1        : 94.431%



Test Data
accuracy  : 92.434%
precision : 98.972%
recall    : 85.556%
f1        : 91.776%
```

4c. Random Forest Prediction

```
In [152]: model_name = 'RandomForest'
          predictions[model_name] = RandomForest.predict(pred_df)
```

```
site_exists      = predictions[predictions[model_name] == 1]
site_nexists     = predictions[predictions[model_name] != 1]
num_site_exists  = site_exists[model_name].count()
num_site_nexists = site_nexists[model_name].count()

print 'Sites exists         : %d ' % (num_site_exists)
print 'Sites does not exists: %d ' % (num_site_nexists)
print 'Percentage of sites predicted having an archeological site: %.3f%s' % \
(100.0*float(num_site_exists)/float(num_site_exists+num_site_nexists), '%')

# plot the map
plotPredictedFromDF(raw_site_df,
                    predictions,
                    model_name=model_name,
                    country='Ireland',
                    resolution='i',
                    alpha_predicted=.05,
                    alpha_common=0.3)
```

```
Sites exists         : 4153
Sites does not exists: 654784
Percentage of sites predicted having an archeological site: 0.630%
```



3.0.5 5. Gradient Boosting Classification

5a. Training the Gradient Boosting Classification Model

In [138]:
```
from sklearn import ensemble

GradientBoosting = ensemble.GradientBoostingClassifier()

start_time = time.time()
GradientBoosting.fit(train_data, train_target)
print "Time to Train: %f seconds" % (time.time() - start_time)
```

```
Time to Train: 0.303152 seconds
```

5b. Testing the Gradient Boosting Classifier

In [139]:
```
from sklearn import metrics

model = GradientBoosting
"""
```

```
    Fit the training data
    and observe the metrics
    """
    preds = model.predict(train_data)
    targs = train_target
    args = (targs, preds)
    print "Training Data"
    print "accuracy  : %.03f%s" % (metrics.accuracy_score(*args)*100, "%")
    print "precision : %.03f%s" % (metrics.precision_score(*args)*100, "%")
    print "recall    : %.03f%s" % (metrics.recall_score(*args)*100, "%")
    print "f1        : %.03f%s" % (metrics.f1_score(*args)*100, "%")

    """
    Fit the test data
    and observe the metrics
    """
    preds = model.predict(test_data)
    targs = test_target
    args = (targs, preds)
    print "\n"
    print "Test Data"
    print "accuracy  : %.03f%s" % (metrics.accuracy_score(*args)*100, "%")
    print "precision : %.03f%s" % (metrics.precision_score(*args)*100, "%")
    print "recall    : %.03f%s" % (metrics.recall_score(*args)*100, "%")
    print "f1        : %.03f%s" % (metrics.f1_score(*args)*100, "%")
```

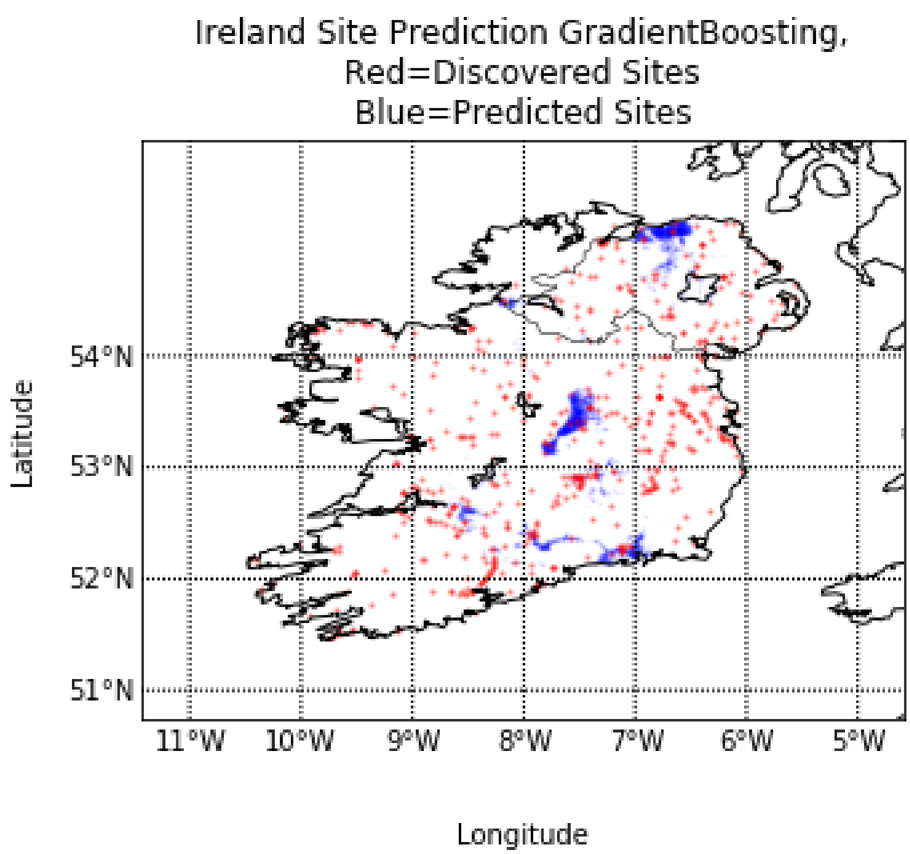
```
Training Data
accuracy  : 97.460%
precision : 99.804%
recall    : 95.136%
f1        : 97.414%


Test Data
accuracy  : 94.079%
precision : 97.826%
recall    : 90.000%
f1        : 93.750%
```

5c. Gradient Boosting Classifier Prediction

```
In [153]: model_name = 'GradientBoosting'
          predictions[model_name] = GradientBoosting.predict(pred_df)

          site_exists      = predictions[predictions[model_name] == 1]
          site_nexists     = predictions[predictions[model_name] != 1]
          num_site_exists  = site_exists[model_name].count()
```

```
num_site_nexists = site_nexists[model_name].count()

print 'Sites exists         : %d ' % (num_site_exists)
print 'Sites does not exists: %d ' % (num_site_nexists)
print 'Percentage of sites predicted having an archeological site: %.3f%s' % \
(100.0*float(num_site_exists)/float(num_site_exists+num_site_nexists), '%')

# plot the map
plotPredictedFromDF(raw_site_df,
                    predictions,
                    model_name=model_name,
                    country='Ireland',
                    resolution='i',
                    alpha_predicted=.01,
                    alpha_common=0.3)
```

```
Sites exists         : 13569
Sites does not exists: 645368
Percentage of sites predicted having an archeological site: 2.059%
```



3.0.6 6. Ada Boosting Classification

6a. Training the Adaptive Boosting Classification Model

```
In [141]: from sklearn import ensemble

          AdaBoosting = ensemble.AdaBoostClassifier()

          start_time = time.time()
          AdaBoosting.fit(train_data, train_target)
          print "Time to Train: %f seconds" % (time.time() - start_time)
```

```
Time to Train: 0.300660 seconds
```

6b. Testing the Ada Boosting Classification

```
In [142]: from sklearn import metrics

          model = AdaBoosting
          """
          Fit the training data
          and observe the metrics
          """
          preds = model.predict(train_data)
          targs = train_target
          args = (targs, preds)
          print "Training Data"
          print "accuracy  : %.03f%s" % (metrics.accuracy_score(*args)*100, "%")
          print "precision : %.03f%s" % (metrics.precision_score(*args)*100, "%")
          print "recall    : %.03f%s" % (metrics.recall_score(*args)*100, "%")
          print "f1        : %.03f%s" % (metrics.f1_score(*args)*100, "%")

          """
          Fit the test data
          and observe the metrics
          """
          preds = model.predict(test_data)
          targs = test_target
          args = (targs, preds)
          print "\n"
          print "Test Data"
          print "accuracy  : %.03f%s" % (metrics.accuracy_score(*args)*100, "%")
          print "precision : %.03f%s" % (metrics.precision_score(*args)*100, "%")
          print "recall    : %.03f%s" % (metrics.recall_score(*args)*100, "%")
          print "f1        : %.03f%s" % (metrics.f1_score(*args)*100, "%")
```

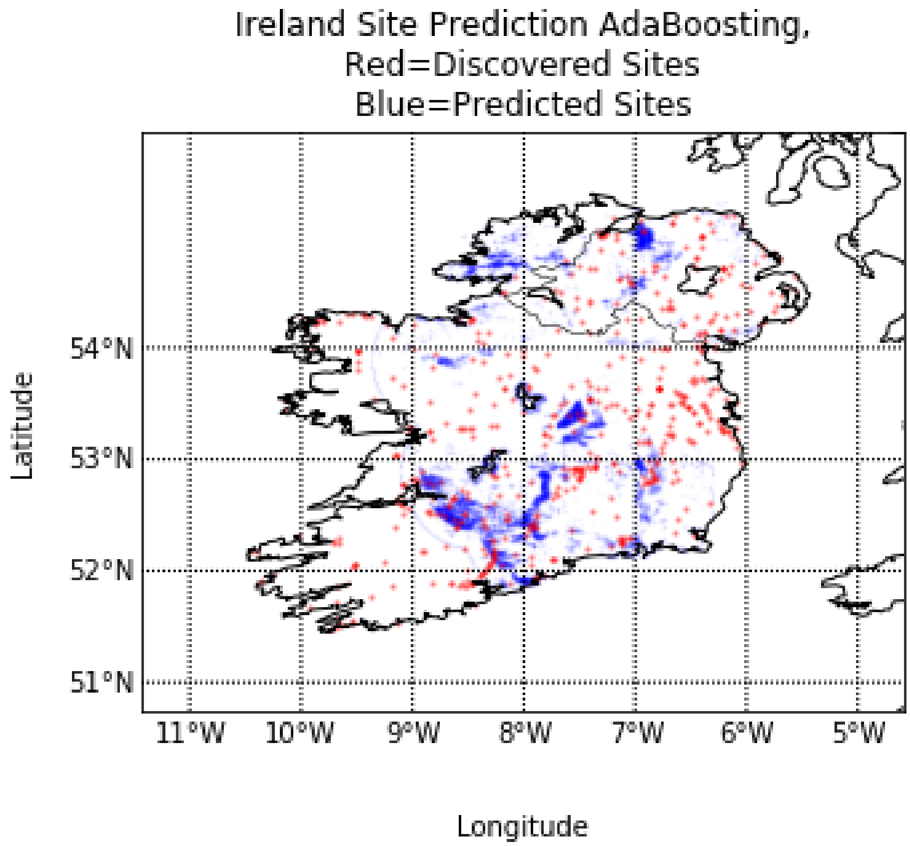
```
Training Data
accuracy  : 94.779%
```

```
precision : 98.189%
recall    : 91.300%
f1        : 94.619%


Test Data
accuracy  : 91.996%
precision : 94.563%
recall    : 88.889%
f1        : 91.638%
```

6c. Ada Boosting Prediction

```
In [154]: model_name = 'AdaBoosting'
          predictions[model_name] = AdaBoosting.predict(pred_df)

          site_exists      = predictions[predictions[model_name] == 1]
          site_nexists     = predictions[predictions[model_name] != 1]
          num_site_exists  = site_exists[model_name].count()
          num_site_nexists = site_nexists[model_name].count()

          print 'Sites exists         : %d ' % (num_site_exists)
          print 'Sites does not exists: %d ' % (num_site_nexists)
          print 'Percentage of sites predicted having an archeological site: %.3f%s' % \
          (100.0*float(num_site_exists)/float(num_site_exists+num_site_nexists), '%')

          # plot the map
          plotPredictedFromDF(raw_site_df,
                              predictions,
                              model_name=model_name,
                              country='Ireland',
                              resolution='i',
                              alpha_predicted=.01,
                              alpha_common=0.3)
```

```
Sites exists         : 34842
Sites does not exists: 624095
Percentage of sites predicted having an archeological site: 5.288%
```


Ireland Site Prediction AdaBoosting,
Red=Discovered Sites
Blue=Predicted Sites
54°N
53°N
52°N
51°N
Latitude
11°W
10°W
9°W
8°W
7°W
6°W
5°W
Longitude