A Machine Learning Approach to Predict Forest Fires using Meteorological Data

December 13, 2018

0.1 BIOSTAT 273: Final Project

0.1.1 Chad Pickering | 12.14.2018

0.1.2 Introduction.

Spatial, temporal, and weather-related data and indexes were collected from the Montesinho Natural Park in northeastern Portugal between 2000 and 2003 (http://www.dsi.uminho.pt/~pcortez/forestfires/). 517 forest fires were registered and recorded during that time interval and are included in this dataset.

Here, we consider ML techniques adapted for a continuous response variable. We will predict the burn area in forest hectares (ha) (1 ha = $10000m^2$) using a decision tree regressor, random forest regressor, and support vector regressor. I wanted to implement some Python-based regression models in addition to the more straightforward classification procedures provided in the accompanying R Markdown document.

0.1.3 Data Pre-Processing.

First, import libraries and preview the structure and content of the data.

```
In [146]: import numpy as np
          import pandas as pd
          import matplotlib.pyplot as plt
          import seaborn as sns
          from sklearn.model_selection import train_test_split
          from sklearn.preprocessing import OneHotEncoder
          from sklearn.preprocessing import LabelEncoder
          from sklearn.svm import SVR
          from sklearn.preprocessing import StandardScaler
          from sklearn.model_selection import GridSearchCV
          from sklearn.tree import DecisionTreeRegressor
          from sklearn.ensemble import RandomForestRegressor

          %matplotlib inline
```

```
In [147]: fires = pd.read_csv('forestfires.csv')
          fires.head(3)
```

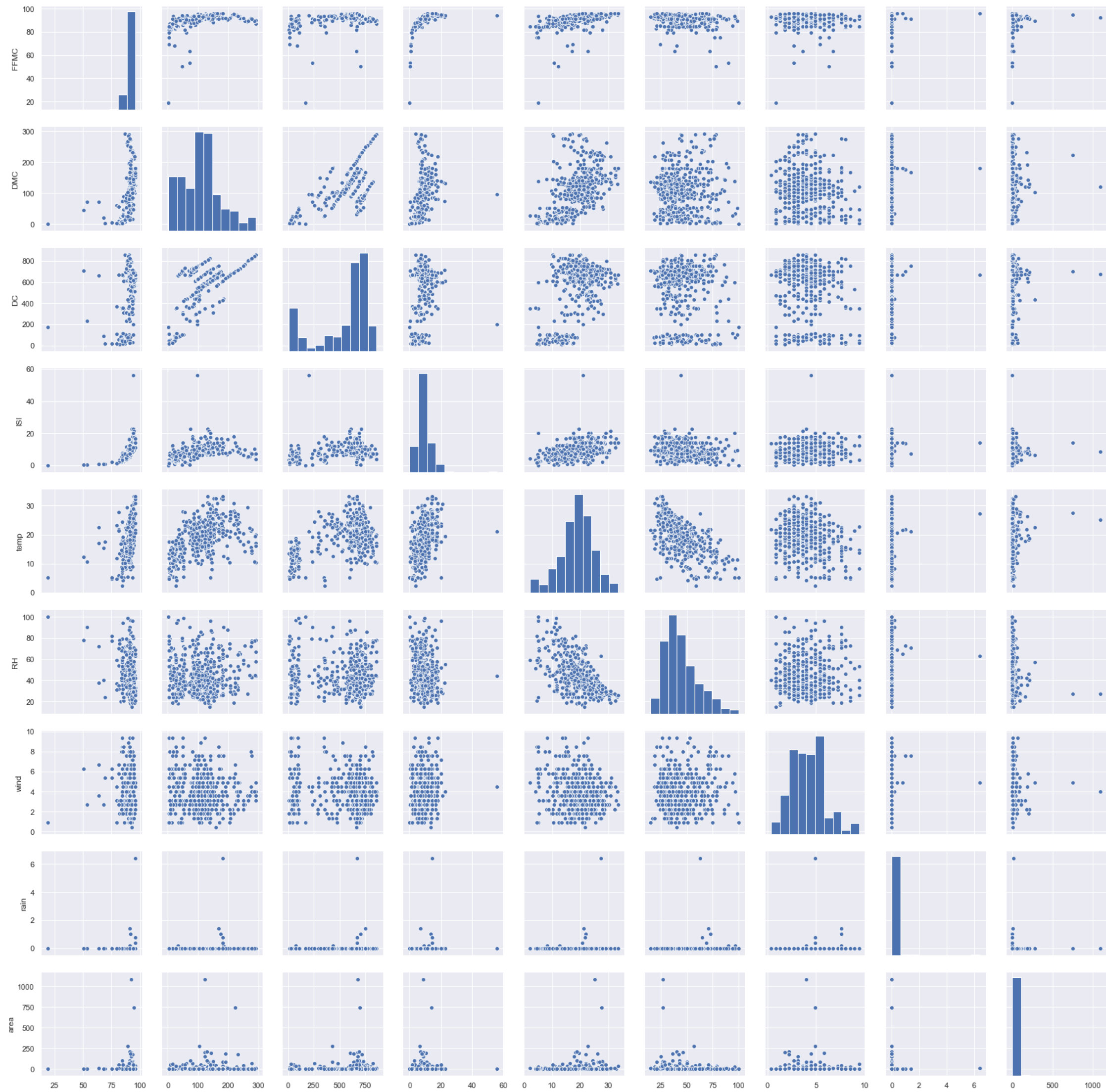
```
Out[147]:    X  Y month  day  FFMC   DMC     DC  ISI  temp  RH  wind  rain  area
          0  7  5   mar  fri  86.2  26.2   94.3  5.1   8.2  51   6.7   0.0   0.0
          1  7  4   oct  tue  90.6  35.4  669.1  6.7  18.0  33   0.9   0.0   0.0
          2  7  4   oct  sat  90.6  43.7  686.9  6.7  14.6  33   1.3   0.0   0.0
```

```
In [148]: fires.describe()
```

```
Out[148]:                 X           Y        FFMC         DMC          DC         ISI  \
          count  517.000000  517.000000  517.000000  517.000000  517.000000  517.000000
          mean     4.669246    4.299807   90.644681  110.872340  547.940039    9.021663
          std      2.313778    1.229900    5.520111   64.046482  248.066192    4.559477
          min      1.000000    2.000000   18.700000    1.100000    7.900000    0.000000
          25%      3.000000    4.000000   90.200000   68.600000  437.700000    6.500000
          50%      4.000000    4.000000   91.600000  108.300000  664.200000    8.400000
          75%      7.000000    5.000000   92.900000  142.400000  713.900000   10.800000
          max      9.000000    9.000000   96.200000  291.300000  860.600000   56.100000

                       temp          RH        wind        rain         area
          count  517.000000  517.000000  517.000000  517.000000   517.000000
          mean    18.889168   44.288201    4.017602    0.021663    12.847292
          std      5.806625   16.317469    1.791653    0.295959    63.655818
          min      2.200000   15.000000    0.400000    0.000000     0.000000
          25%     15.500000   33.000000    2.700000    0.000000     0.000000
          50%     19.300000   42.000000    4.000000    0.000000     0.520000
          75%     22.800000   53.000000    4.900000    0.000000     6.570000
          max     33.300000  100.000000    9.400000    6.400000  1090.840000
```

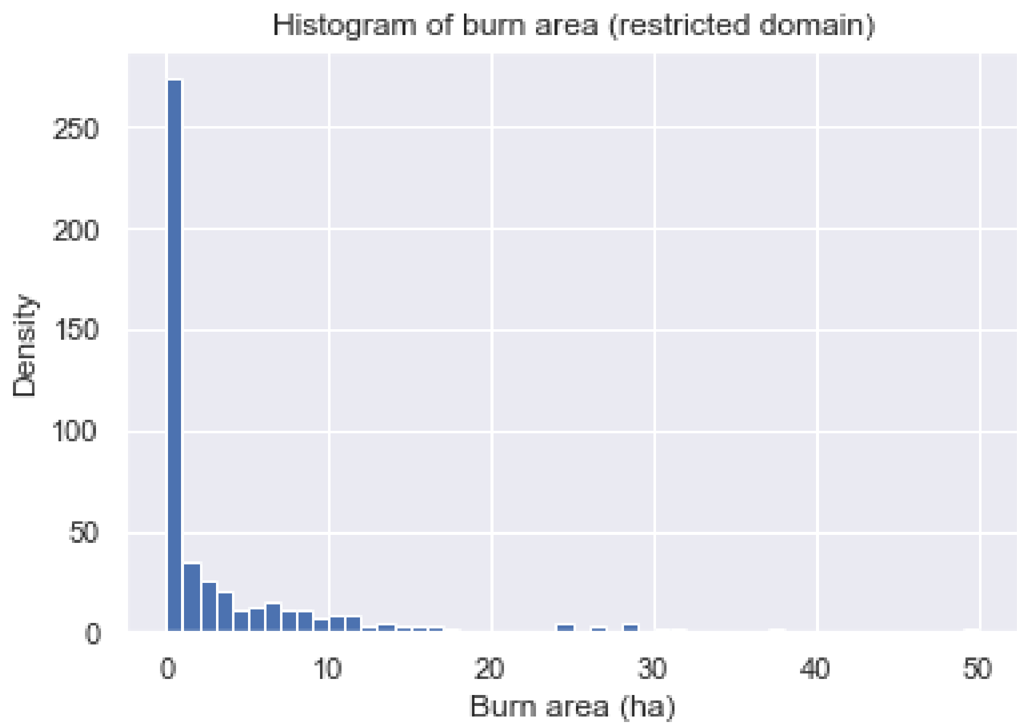
```
In [149]: sns.set()
          sns.pairplot(fires[['FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', 'wind', 'rain', 'area']
```

```
Out[149]: <seaborn.axisgrid.PairGrid at 0x125468b90>
```



Upon exploring the response variable, burn area, we notice that the distribution is highly left skewed, with about half of the fires (247/517) having a burn area of <0.01 hectare, which is represented as 0.

```
In [150]: plt.hist(fires['area'], bins=50, range=[0, 50])
          plt.ylabel('Density')
          plt.xlabel('Burn area (ha)')
          plt.title('Histogram of burn area (restricted domain)')

Out[150]: Text(0.5,1,'Histogram of burn area (restricted domain)')
```

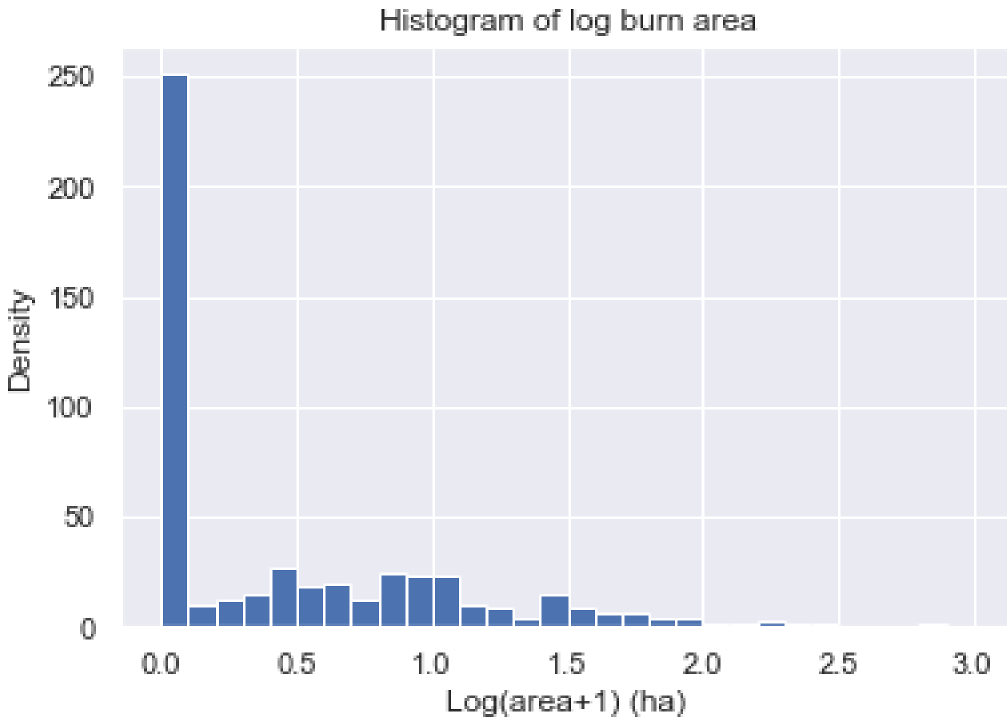


The large magnitude of small fires in the dataset leads us to propose a logarithmic transformation of the response variable: $Y_{transf} = log(Y + 1)$. Adding 1 prevents infinite response values. Note that I make a log base 10 transform here, for convenience purposes. The base does not change the outcome, but makes interpretation easier.

```
In [151]: fires['logarea'] = np.log10(fires['area']+1)
```

```
In [152]: plt.hist(fires['logarea'], bins=30, range=[0, 3])
          plt.ylabel('Density')
          plt.xlabel('Log(area+1) (ha)')
          plt.title('Histogram of log burn area')
```

```
Out[152]: Text(0.5,1,'Histogram of log burn area')
```



The frequencies of fires are quite sporadic and uneven through the months of the year, so it is a good idea to combine months into seasons instead.

In [153]:
```
def month_season (df):
    if df['month'] in ['jan','feb','mar']:
        return 'winter'
    if df['month'] in ['apr','may','jun']:
        return 'spring'
    if df['month'] in ['jul','aug','sep']:
        return 'summer'
    if df['month'] in ['oct','nov','dec']:
        return 'fall'

fires['season'] = fires.apply (lambda df: month_season (df), axis=1)
```

In [154]:
```
fires.head(3)
```

Out[154]:
```
   X  Y month  day  FFMC   DMC     DC  ISI  temp  RH  wind  rain  area  \
0  7  5   mar  fri  86.2  26.2   94.3  5.1   8.2  51   6.7   0.0   0.0
1  7  4   oct  tue  90.6  35.4  669.1  6.7  18.0  33   0.9   0.0   0.0
2  7  4   oct  sat  90.6  43.7  686.9  6.7  14.6  33   1.3   0.0   0.0

   logarea  season
0      0.0  winter
1      0.0    fall
2      0.0    fall
```

Now we should encode the season and day of the week as values for the coming procedures. Encoded values follow the original variable's levels in alphabetical order.

```
In [155]: enc = LabelEncoder()
          enc.fit(fires['season'])
          fires['season_enc'] = enc.transform(fires['season'])
          enc.fit(fires['day'])
          fires['day_enc'] = enc.transform(fires['day'])
          fires.head()
```

```
Out[155]:    X  Y month  day  FFMC   DMC     DC  ISI  temp  RH  wind  rain  area  \
          0  7  5   mar  fri  86.2  26.2   94.3  5.1   8.2  51   6.7   0.0   0.0
          1  7  4   oct  tue  90.6  35.4  669.1  6.7  18.0  33   0.9   0.0   0.0
          2  7  4   oct  sat  90.6  43.7  686.9  6.7  14.6  33   1.3   0.0   0.0
          3  8  6   mar  fri  91.7  33.3   77.5  9.0   8.3  97   4.0   0.2   0.0
          4  8  6   mar  sun  89.3  51.3  102.2  9.6  11.4  99   1.8   0.0   0.0

             logarea  season  season_enc  day_enc
          0      0.0  winter           3        0
          1      0.0    fall           0        5
          2      0.0    fall           0        2
          3      0.0  winter           3        0
          4      0.0  winter           3        3
```

0.1.4 Split data into training and test sets.

```
In [156]: X_data = fires.drop(['area','logarea','month','day','season'], axis=1)
          y_data = fires['logarea']
```

```
In [157]: test_size = 0.4 # training set 70%; test set 30%
          X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=test_s
```

```
In [158]: y_train_orig = y_train # save copy of vector
          b = y_train.values.reshape(y_train.size,1)
          y_train = pd.DataFrame(b)
          y_train.columns = ['logarea']
```

```
In [159]: y_test_orig = y_test # save copy of vector
          b2 = y_test.values.reshape(y_test.size,1)
          y_test = pd.DataFrame(b2)
          y_test.columns = ['logarea']
```

0.1.5 Regression Error Characteristic (REC) Curve Estimation Function.

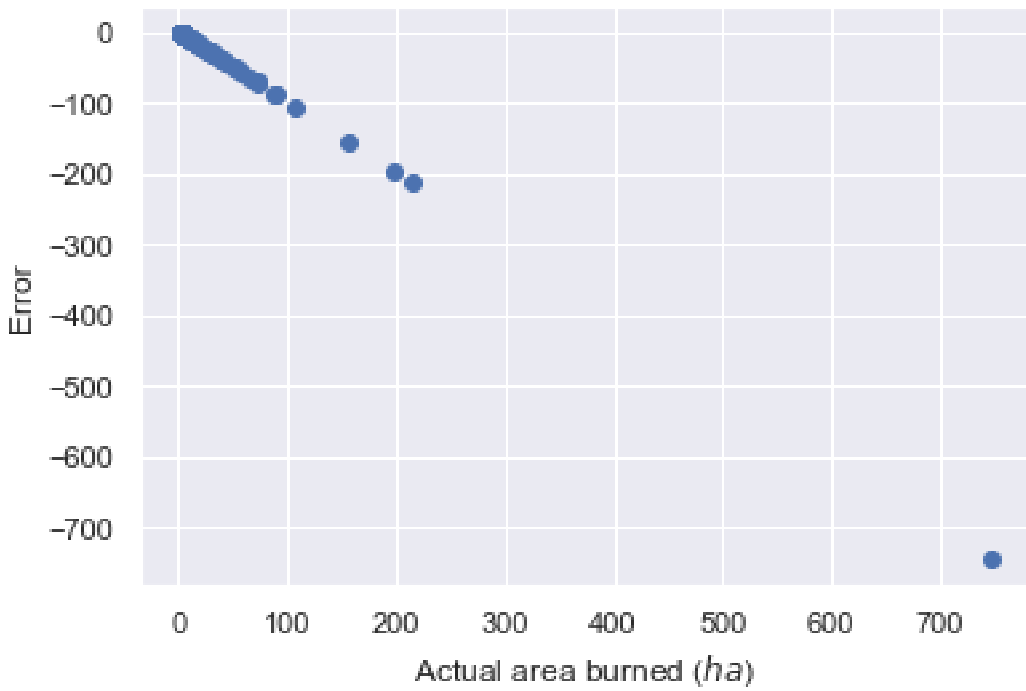
While Receiver Operating Characteristic (ROC) curves allow the visualization and comparison of classification results, Regression Error Characteristic (REC) curves generalize these to regression techniques. REC curves plot the error tolerance on the domain versus the fraction of points predicted within the tolerance on the y-axis. The resulting curve estimates the cumulative distribution function of the error. An ideal regressor would yield an REC as close to 1 as possible.

```
In [160]: def rec(m,n,tol):
              if type(m)!='numpy.ndarray':
                  m=np.array(m)
              if type(n)!='numpy.ndarray':
                  n=np.array(n)
              l=m.size
              percent = 0
              for i in range(l):
                  if np.abs(10**m[i]-10**n[i])<=tol:
                      percent+=1
              return 100*(float(percent)/l)

          tol_max = 20 # max tolerance limit for REC curve x-axis; abs value of error in the p
          y_test_array = np.array(y_test_orig.values) # make np.array
```

0.1.6 Support Vector Regressor via GridSearch.

The scikitlearn GridSearchCV package is able to try a variety of combinations of parameters to input into ML algorithms to see what works best. GridSearchCV takes in a dictionary as input that describes such parameters and a model on which to train.

```
In [161]: scaler = StandardScaler()

          # Parameter grid for the Grid Search
          param_grid = {'C': [0.01,0.1,1,10], 'epsilon': [10,1,0.1,0.01,0.001,0.0001], 'kernel

In [162]: grid_SVR = GridSearchCV(SVR(), param_grid, refit=True, verbose=0, cv=5) # 5-fold cro
          grid_SVR.fit(scaler.fit_transform(X_train), scaler.fit_transform(y_train))
          print("Best parameters obtained by GridSearch:", grid_SVR.best_params_)

('Best parameters obtained by GridSearch:', {'epsilon': 1, 'C': 0.1, 'kernel': 'rbf'})


In [163]: a_svr = grid_SVR.predict(X_test)
          print("RMSE for Support Vector Regression:", np.sqrt(np.mean((y_test_orig-a_svr)**2))
          a_svr = np.array(a_svr) # np.ndarray type

('RMSE for Support Vector Regression:', 0.69115705177856)


In [164]: plt.xlabel("Actual area burned ($ha$)")
          plt.ylabel("Error")
          plt.grid(True)
          plt.scatter(10**(y_test_orig),10**(a_svr)-10**(y_test_orig))

Out[164]: <matplotlib.collections.PathCollection at 0x129015c10>
```



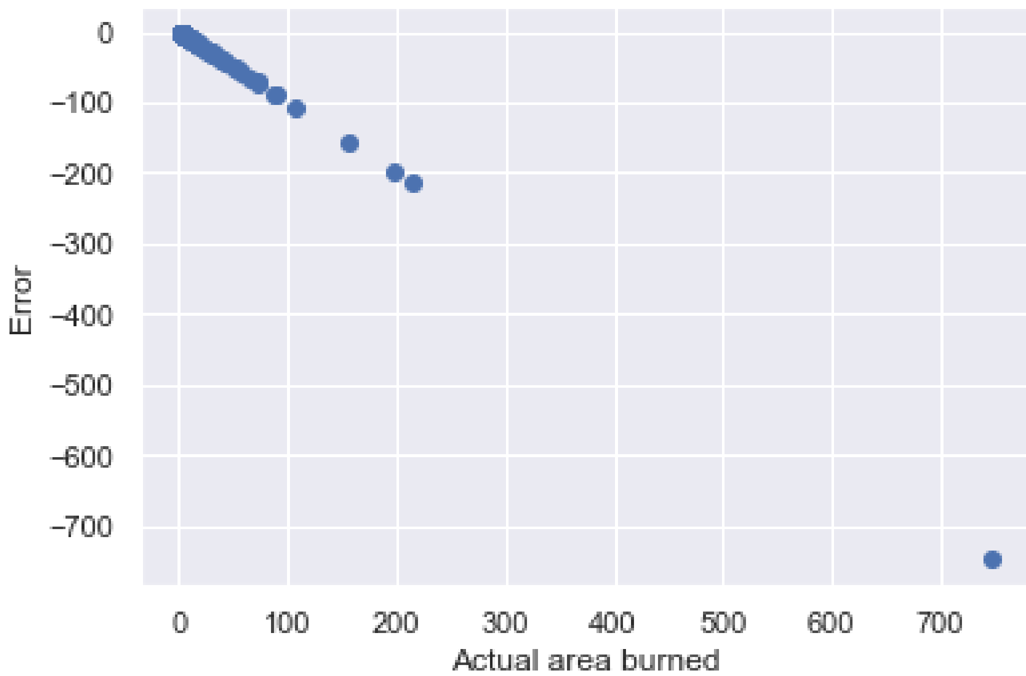
0.1.7 Decision Tree Regressor

```
In [165]: tree_model = DecisionTreeRegressor(max_depth=10, criterion='mae')
          tree_model.fit(scaler.fit_transform(X_train), scaler.fit_transform(y_train))

          a_dt = tree_model.predict(X_test)
          print("RMSE for Decision Tree:",np.sqrt(np.mean((y_test_orig-a_dt)**2)))
          a_dt = np.array(a_dt) # np.ndarray type
```

```
('RMSE for Decision Tree:', 1.44498470615323)
```

```
In [166]: plt.xlabel("Actual area burned")
          plt.ylabel("Error")
          plt.grid(True)
          plt.scatter(10**(y_test_orig),10**(a_dt)-10**(y_test_orig))
```

```
Out[166]: <matplotlib.collections.PathCollection at 0x12908c650>
```



0.1.8 Random Forest Regressor

```
In [167]: param_grid = {'max_depth': [5,10,15,20,50], 'max_leaf_nodes': [2,5,10], 'min_samples_
                        'min_samples_split':[2,5,10]}
          grid_RF = GridSearchCV(RandomForestRegressor(), param_grid, refit=True, verbose=0, cv
          grid_RF.fit(X_train, y_train)

          print("Best parameters obtained by Grid Search:", grid_RF.best_params_)
```

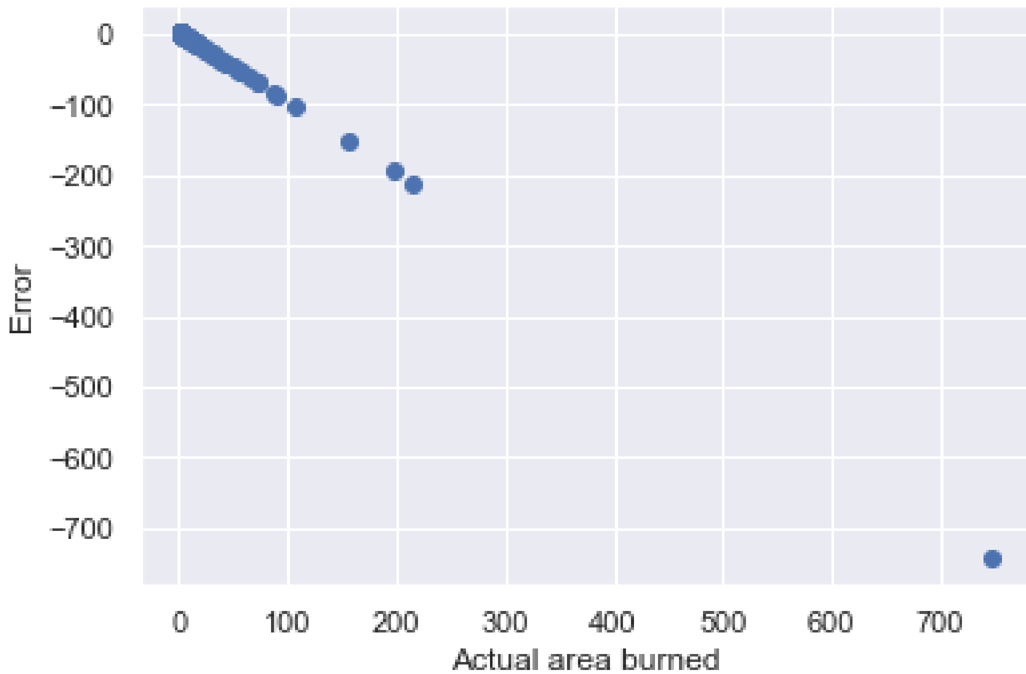
```
('Best parameters obtained by Grid Search:', {'min_samples_split': 5, 'max_leaf_nodes': 2, 'ma
```

```
In [168]: a_rf = grid_RF.predict(X_test)
          rmse_rf = np.sqrt(np.mean((y_test_orig-a_rf)**2))
          print("RMSE for Random Forest:", rmse_rf)
          a_rf = np.array(a_rf) # np.ndarray type
```

```
('RMSE for Random Forest:', 0.6350727990651835)
```

```
In [169]: plt.xlabel("Actual area burned")
          plt.ylabel("Error")
          plt.grid(True)
          plt.scatter(10**(y_test_orig), 10**(a_rf)-10**(y_test_orig))
```

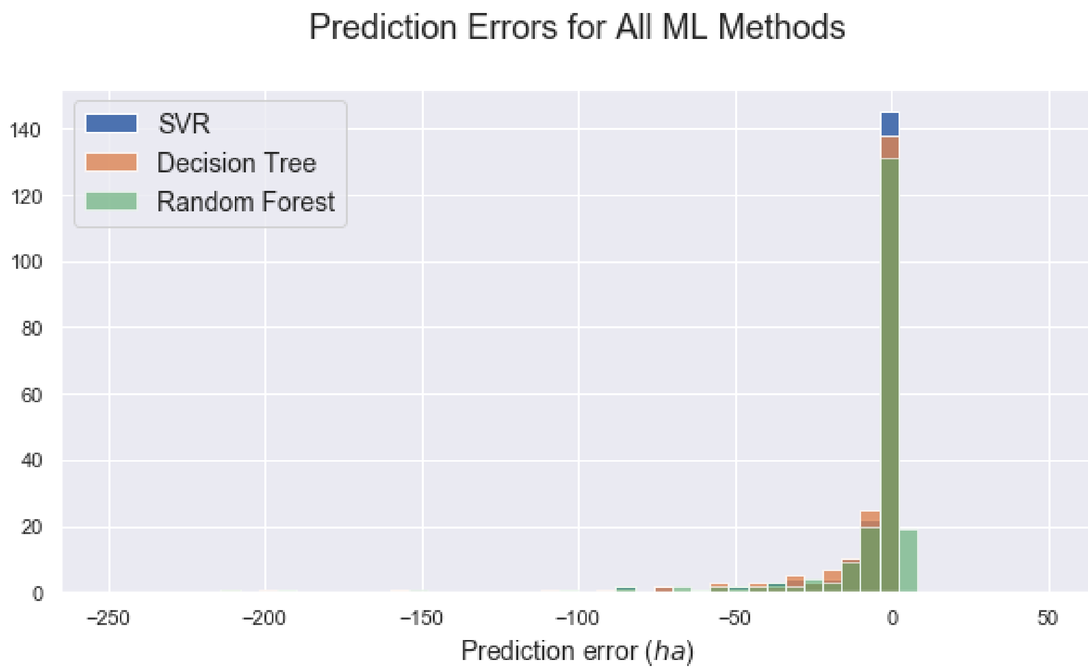
Out[169]: <matplotlib.collections.PathCollection at 0x1293206d0>



0.1.9 Prediction Error Patterns

In [170]:
```
plt.figure(figsize=(10, 5))
plt.title("Prediction Errors for All ML Methods\n", fontsize=18)
plt.xlabel("Prediction error ($ha$)", fontsize=14)
plt.grid(True)
plt.hist(10**(a_svr.reshape(a_svr.size,))-10**(y_test_orig), bins=50, range=[-250, 50
plt.hist(10**(a_dt.reshape(a_dt.size,))-10**(y_test_orig), bins=50, range=[-250, 50]
plt.hist(10**(a_rf.reshape(a_rf.size,))-10**(y_test_orig), bins=50, range=[-250, 50]
plt.legend(['SVR', 'Decision Tree', 'Random Forest'], fontsize=14)
```

Out[170]: <matplotlib.legend.Legend at 0x12949fd90>



For all three ML methods, moderate to large fires tend to get predicted as being smaller than they truly were. We can see, thanks to the opacity in the following plot, that the SVR has a higher rate of very small errors - this suggests an overall better performance for smaller fires.

0.1.10 REC Curve Comparison and Conclusions

```
In [171]: rec_SVR=[]
          for i in range(tol_max):
              rec_SVR.append(rec(a_svr,y_test_array,i))
```

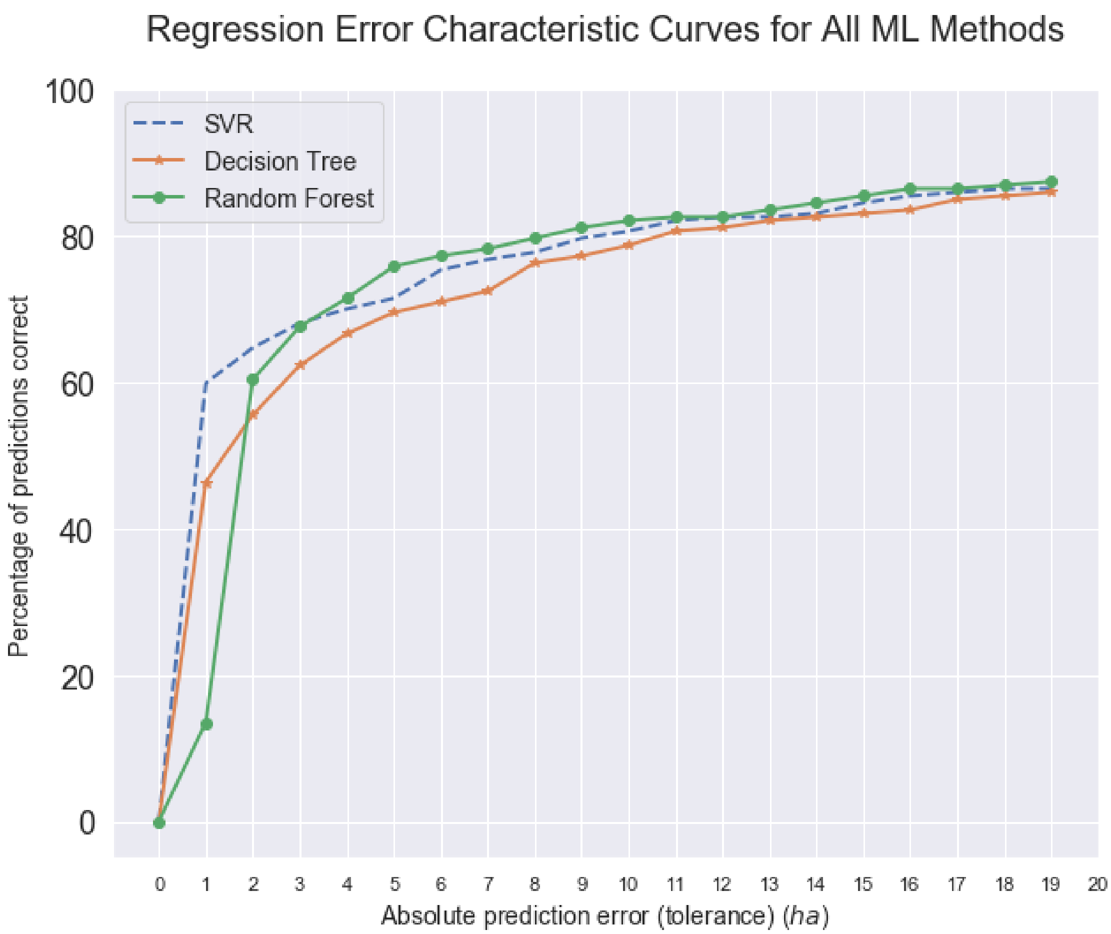
```
In [172]: rec_DT=[]
          for i in range(tol_max):
              rec_DT.append(rec(a_dt,y_test_array,i))
```

```
In [173]: rec_RF=[]
          for i in range(tol_max):
              rec_RF.append(rec(a_rf,y_test_array,i))
```

```
In [174]: plt.figure(figsize=(10, 8))
          plt.title("Regression Error Characteristic Curves for All ML Methods\n", fontsize=20)
          plt.xlabel("Absolute prediction error (tolerance) ($ha$)", fontsize=14)
          plt.ylabel("Percentage of predictions correct", fontsize=14)
          plt.xticks([i for i in range(0, tol_max+1, 1)], fontsize=11)
          plt.yticks([i*20 for i in range(6)], fontsize=18)
          plt.xlim(-1, tol_max)
          plt.ylim(-5, 100)
          plt.grid(True)
```

```
plt.plot(range(tol_max), rec_SVR, '--', lw=2)
plt.plot(range(tol_max), rec_DT, '*-', lw=2)
plt.plot(range(tol_max), rec_RF, 'o-', lw=2)
plt.legend(['SVR', 'Decision Tree', 'Random Forest'], fontsize=14)
```

```
Out[174]: <matplotlib.legend.Legend at 0x1295f2d50>
```



The support vector regressor typically does a much better job of predicting small fires than the decision tree or random forest regressors. It is worth noting that moderate sized fires are typically predicted at higher accuracy by the random forest regressor, but all three ML methods seem to essentially converge when absolute prediction error/tolerance exceeds 15. At times, depending on the assignment of anomalous large fires to training and test sets, the decision tree REC curve yields a very low AOC - its unstable predictive power makes this method the least desirable of the bunch; although, to its credit, on its best runs it performs comparably, or even slightly superior, to the other two for large tolerance values.