19. Tree Ensembles

Bruce E. Shapiro

Getting Started in Machine Learning

Last revised: March 31, 2019

Ensembles of Trees

- Compare the results of many decision trees.
- `sklearn.ensemble` library
  - **Boosting**. Computes error; iterates and finds weighted average.
    - **Adaptive Boosting** (Adaboost) - modifies weights of sample
    - **Gradient Boosting** - minimizes errors by gradient descent
  - **Bagging** or **bootstrap aggregation**. Generates multiple training sets and forms weighted average.
  - **Random Forests**. Uses a subset of features in each subset.
  - **Isolation Forests**. Variant of random forest that computes path length, thereby identifying outliers (shorter paths).
  - ...
-

Wine data file

```
fixed volatile  citric  residual chlor- free    total   density pH      sufph-  alcohol quality
acid- acidity   acid    sugar   ides    sulfur  sulfur
ity                                     dioxide dioxide
7.0     0.27    0.36    20.7    0.045   45.0    170.0   1.0010  3.00    0.45    8.8     6
6.3     0.30    0.34    1.6     0.049   14.0    132.0   0.9940  3.30    0.49    9.5     6
8.1     0.28    0.40    6.9     0.050   30.0    97.0    0.9951  3.26    0.44    10.1    6
                .
                .
                .
                (etc)
```

Read File, Create `X`, `Y` Arrays

```
import pandas as pd
import numpy as np
white=pd.read_csv("https://archive.ics.uci.edu/ml/
    machine-learning-databases/wine-quality/
    winequality-white.csv", sep=";")
Y=np.array(white["quality"])
X=np.array(white)[:,:-1]
```

Forests and Trees: Approach

- for each forest size $n = 2, \ldots$:
  - repeat N times:
    - perform a train/test split
    - calculate the classification error
  - calculate:
    - $\mu =$ mean of N classification errors
    - $\sigma =$ standard deviation of N classification errors
- plot $\mu \pm \sigma$ vs forest size n (number of trees)

Code to Do **nruns** Train/Test sSlits

- let **clf** is a classifier from **sklearn**:
- Run **nruns** train/test split

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
def evaluate_classifier(clf, X, Y, nruns):
    errs=[]
    for j in range(nruns):
        XTRAIN, XTEST, YTRAIN, YTEST=\
           train_test_split(X,Y)
        clf.fit(XTRAIN, YTRAIN)
        YP=clf.predict(XTEST)
        errs.append(1-accuracy_score(YTEST,YP))
    return(np.mean(errs), np.std(errs))
```

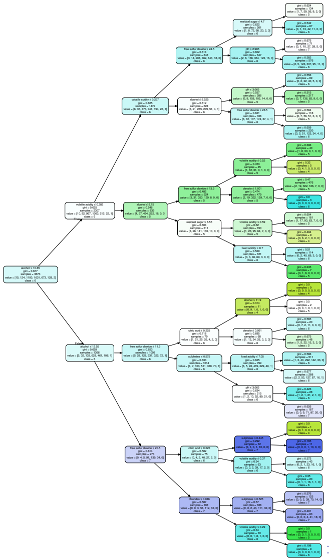
- Return value is (μ, σ)

Baselines: Single Tree (no ensemble)

```
DT5=DecisionTreeClassifier(max_depth=5)
DTT5mu,DTT5sig=evaluate_classifier(DT5,X,Y,25)
DT=DecisionTreeClassifier(max_depth=None)
DTFullmu, DTFullsig=evaluate_classifier(DT,X,Y,25)
print("Max Depth = 5: ", round(DTT5mu,3),
   round(DTT5sig,3))
print("Max Depth = No Limit: ", round(DTFullmu,3),
   round(DTFullsig,3))
```

```
Max Depth = 5:   0.47 0.015
Max Depth = No Limit:  0.407 0.015
```

Depth-5 Decision Tree



Decision Tree - No Depth Limit

Full tree at `https://github.com/biomathman/machine-learning-in-python-book/blob/master/extras/white-wine-class.pdf`

Evaluation of Classifier vs. Forest Size

- Let **`clf`** be an sklearn ensemble classifier
- Let **`forestsizes`** be a list of numbers of trees in each forest
- Let **`repeats`** be the # of train/test repeats for each forest

```
def generate_forests(clf, X, Y, repeats, forestsizes):
    results=[]
    for ntrees in forestsizes:
        classifier=clf(n_estimators=ntrees)
        mu,sigma=evaluate_classifier(classifier,X,Y,repeat
        results.append([mu,sigma])
        print(ntrees,mu,sigma)
    mus=np.array(results).T[0]
    sigs=np.array(results).T[1]
    return(mus,sigs)
```

- Return value is tuple of lists $([\mu_1, \mu_2, ...], [\sigma_1, \sigma_2, ...])$

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
forest_sizes=[2,3,4,5, 7,10,20,30,40,50,70,100,300,500]
BoostMeans, BoostSigmas=\
  generate_forests(GradientBoostingClassifier,X,Y,100,\
  forest_sizes)
```

```
2 0.47875102040816325 0.013943399597534543
3 0.47075102040816325 0.013394749292965804
4 0.466432653061224 5 0.01310593368323248 1
5 0.46349387755102034 0.01186994917685235
7 0.4559428571428571 0.01418974411270947 4
10 0.45537142857142 87 0.012687014353050224
20 0.44059591836734696 0.011396956271079483
30 0.43239183673469 39 0.01581731609807552
40 0.42953469387755105 0.011709309901058097
50 0.42402448979591834 0.014190589416523132
70 0.41570612244897 96 0.014030428050738999
100 0.40977959183673 47 0.013007020178639313
300 0.3734857142857142 0.012996513164524244
500 0.3604000000000005 0.013970156804161556
```

Bagging

```
from sklearn.ensemble import BaggingClassifier
BAGmu, BAGsig=\
  generate_forests(BaggingClassifier,X,Y,100,forest_sizes)
```

```
2 0.44858775510204085 0.0144558867383183
3 0.42013877551020407 0.013197278631040045
4 0.3969551020408163 0.014616966515689327
5 0.38026122448979593 0.01435592310209574
7 0.3678612244897959 0.013513424611524709
10 0.3537795918367347 0.012432725420883669
20 0.34132244897959174 0.011983762441976329
30 0.33408979591836735 0.012010613104889854
40 0.33131428571428573 0.011242816948816986
50 0.3295183673469388 0.011165616287904261
70 0.3303591836734694 0.012516016228180335
100 0.32861224489795915 0.011283259872606404
300 0.32581224489795924 0.01089866542082189
500 0.32310204081632654 0.01112842004111149
```

Random Forests

```
from sklearn.ensemble import RandomForestClassifier
RFMeans, RFSigmas=generate_forests(RandomForestClassifier,
  X,Y,100,forest_sizes)
```

```
2 0.45168979591836733 0.014513119486674303
3 0.4197795918367347 0.015092928304275511
4 0.3940571428571429 0.01423224044397006
5 0.3770040816326531 0.012461247885965058
7 0.3638448979591837 0.01528397635960942
10 0.351526530612245 0.012808150258184626
20 0.3343755102040816 0.013125807168611453
30 0.3298857142857143 0.013076977744829866
40 0.3275591836734694 0.012925117364733224
50 0.3237632653061225 0.013077996884039956
70 0.3228734693877551 0.011163980874726634
100 0.3206448979591836 0.010791759547718657
300 0.31915918367346935 0.012994449204209585
500 0.3174204081632653 0.013213258016927827
```

Relative Feature Importance (1/2)

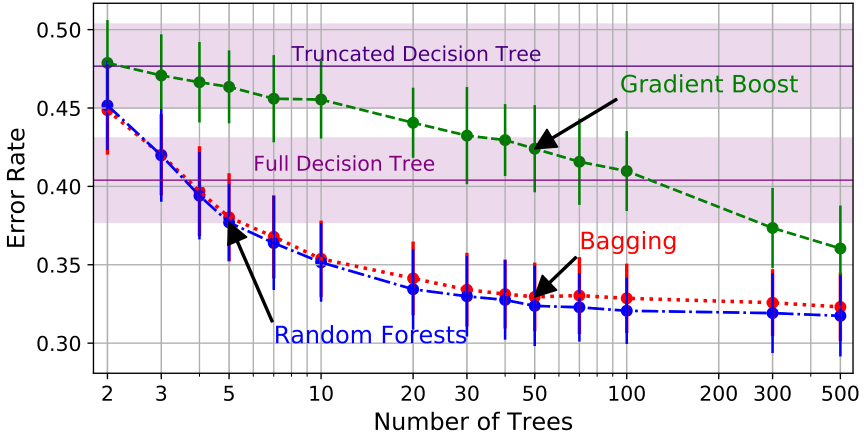
```
print(RF.feature_importances_)
```

```
[0.0750037  0.09881211 0.08138214 0.08818542
 0.08488747 0.09527255 0.09047286 0.10277019
 0.08668786 0.0787098  0.1178159 ]
```

Relative Feature Importance (2/2)

```
names=np.array(white.columns)[:-1].tolist()
importances=RF.feature_importances_.tolist()
imp=list(zip(importances,names))
imp.sort(reverse=True)
for impt,name in imp:
    print("%20s %5.3f" %(name,impt/max(importances)))
```

```
             alcohol 1.000
             density 0.872
    volatile acidity 0.839
 free sulfur dioxide 0.809
total sulfur dioxide 0.768
      residual sugar 0.749
                  pH 0.736
           chlorides 0.721
         citric acid 0.691
           sulphates 0.668
       fixed acidity 0.637
```



Truncated Decision Tree
Gradient Boost
Full Decision Tree
Bagging
Random Forests
Error Rate
0.50
0.45
0.40
0.35
0.30
2
3
5
10
20
30
50
100
200
300
500
Number of Trees

References

1. UCI Machine Learning Repository
   `http://archive.ics.uci.edu/ml`. Irvine, CA: University of California, School of Information and Computer Science.
2. P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.