CS 484
Introduction to Machine Learning

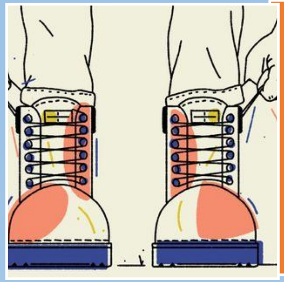
Week 13, April 15, 2021

Spring Semester 2021

ILLINOIS TECH

College of Computing

Week 13 Agenda: Ensemble Learning



Bootstrapping

Bagging

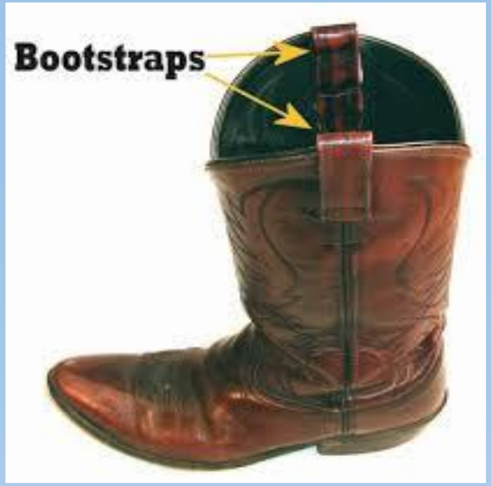
Boosting

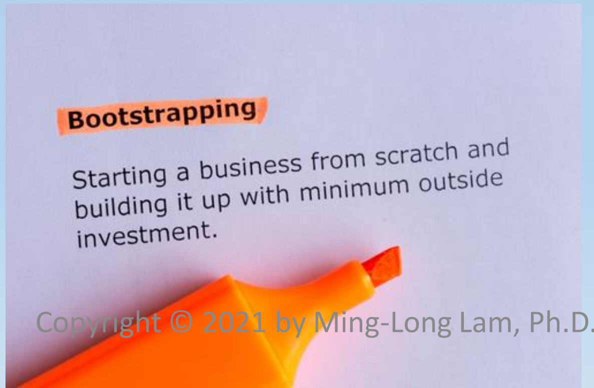
Bootstrap, the Dictionary Meaning

Bootstrap

- (*noun*) A looped strap sewed at the side or the rear top of a boot to help in pulling it on

- (*Idiom*) *To pull oneself up by one's bootstraps* – To succeed only on one's effort or abilities, despite limited resources, or to function independently without outside assistance




Bootstrapping

Starting a business from scratch and building it up with minimum outside investment.

A General Statistical Problem

Observe a random sample of independently and identically distributed (i.i.d.) observations $\mathbf{X} = (X_1, \dots, X_n)$ from a probability distribution F

$\downarrow$

Compute a statistic $s(\mathbf{X})$ based on the observed sample $\mathbf{X}$

$\downarrow$

Estimate the sampling distribution, the 95% confidence interval, or the standard error of the statistic $s(\mathbf{X})$

The Typical Textbook Problem

The probability distribution is $F \sim N(\mu, \sigma^2)$
F is the Normal distribution with mean μ and variance σ^2

The statistic $s(\mathbf{X})$ is the sample mean
$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$

The sample standard deviation
$s = \sqrt{\frac{1}{(n-1)}\sum_{i=1}^{n}(X_i - \bar{X})^2}$

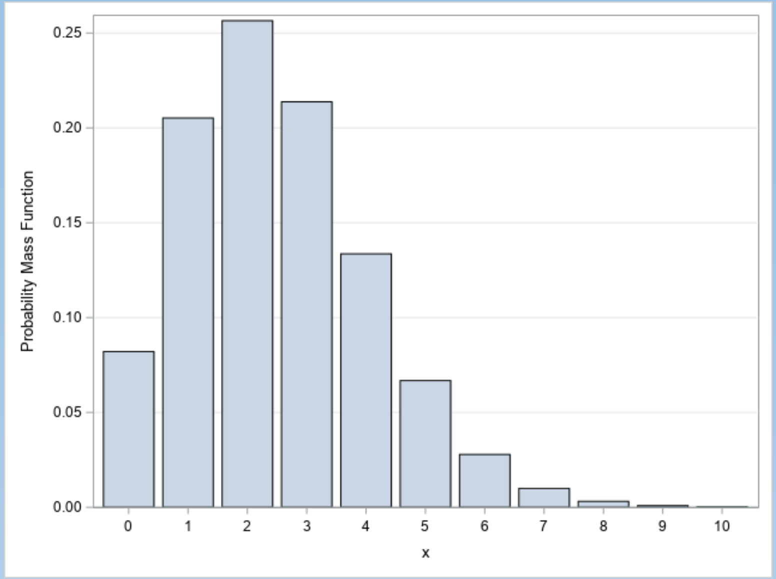
The sampling distribution of mean $\bar{X}$ is $N(\mu, \sigma^2/n)$

The standard error of the mean is $s/\sqrt{n}$

The 95% confidence interval is $(\bar{X} - 1.96\, s/\sqrt{n}, \bar{X} + 1.96\, s/\sqrt{n})$

Deviation from Typical Textbook Problem

What if the distribution F is not normal, say Poisson instead?



The central limit theorem states that if you take sufficiently large random samples from the population with replacement, then the sample means will asymptotically follow a normal distribution.

The Central Limit Theorem asserts that the sample mean will distribute asymptotically as a Normal distribution although the probability distribution F is not Normal.

Deviation from Typical Textbook Problem

What if the statistic is the median, the standard deviation, or the range?

What if my sample is not that big, say, $n = 10$?

What if the distribution F is multivariate and the statistic is the Pearson correlation coefficient?

A Hypothetical Approach

If we can …

- Have full access to the original population
- Repeatedly draw a random sample of size *n* from the original population
- Compute the statistic $s_k(\mathbf{X})$, where $k = 1, 2, \ldots$ are indices to the draws

Then we can …

- Produce a histogram of the array of these statistics
- Calculate the standard deviation of these statistics
- Determine an interval which covers 95% of these statistics

Reality is …

The original population cannot be directly observed

- We may not locate all patients who had a deadly disease but were not diagnosed.
- Some of them may have passed away due to the disease.
- We cannot accurately calculate the mortality rate of the disease.

It is logistically infeasible to observe the original population

- Not everyone U.S. citizens who earn income live in the States.
- The U.S. Census may not able to contact every U.S. citizens.
- Thus, we cannot accurately calculate the U.S. Median Income

Resampling is the Solution

The sample carries the genes of the population because they are from the population

If we ***can introduce*** variations into the sample, then we may create another sample that represents the population.

Pretend the sample is the population, draw with replacement from this *make-believe* population to create another sample that represents the population

Thus simulate as many “samples” as we can handle from the true population

Bootstrap Method

A Brief History

- The bootstrap method was first published by Bradley Efron in 1979
  - Professor of Statistics at Stanford, http://statweb.stanford.edu/~ckirby/brad/
- *Bootstrap Methods: Another Look At The Jackknife*, The Annals of Statistics, 1979, Volume 7, Number 1, Pages 1-26
- *The Jackknife, the Bootstrap and Other Resampling Plans*, 1982, Society for Industrial and Applied Mathematics (SIAM), U.S.
- *An Introduction to the Bootstrap*, 1993, Chapman & Hall, co-authored with R.J. Tibshirani (also at Stanford)

Bootstrap Method

The Simple Algorithm:

1. Construct the empirical probability distribution $\hat{F}$, assigning probability mass $1/n$ to each observation.
2. Repeat the following two tasks as many times as you desire:
   a) Draw a bootstrap sample which is a random sample of size n with replacement from $\hat{F}$.
   b) Compute the statistic: $s(\mathbf{X})$ based on the bootstrap sample.
3. Approximate the sampling distribution of $s(\mathbf{X})$ by the empirical distribution of the statistics based on the bootstrap samples

In the Original Theory …

- DO use the bootstrap method to estimate:
  - Sampling distribution of $s(\mathbf{X})$
  - Standard error of $s(\mathbf{X})$
  - Confidence interval of $s(\mathbf{X})$
  - Significance value of a hypothesis test

- DO NOT use any location estimates (e.g., mean, median and mode) of the sampling distribution of $s(\mathbf{X})$ to "re-estimate" the statistic $s(\mathbf{X})$.
- It is because the statistic $s(\mathbf{X})$ has already been estimated by the original sample data.

Bootstrap Method

Sampling With Replacement Using Python:

```
def sample_wr (inData):
   n = len(inData)
   outData = numpy.empty((n,1))
   for i in range(n):                         1
      j = int(random.random() * n)                          2
      outData[i] = inData[j]                           3
   return outData
```

1. The bootstrap sample should have the same number of observations n as the original sample.
2. Draw a random integer j between 0 and $n-1$.
3. Pull the j^{th} observation from the original sample into the bootstrap sample.

Bootstrap Method

```
import numpy
import random

# Create a bootstrap sample from the population
def sample_wr (inData):
    n = len(inData)
    outData = numpy.empty((n,1))
    for i in range(n):
        j = int(random.random() * n)
        outData[i] = inData[j]
    return outData

x = numpy.array([1,2,3,4,5,6,7,8,10])
unique, counts = numpy.unique(x, return_counts = True)
print('Original Sample:\n', numpy.asarray((unique,
counts)).transpose())

# Check the bootstrap sample
random.seed(20191113)
sample1 = sample_wr(x)
unique, counts = numpy.unique(sample1, return_counts = True)
print('Sample 1:\n', numpy.asarray((unique,
counts)).transpose())

sample2 = sample_wr(x)
unique, counts = numpy.unique(sample2, return_counts = True)
print('Sample 2:\n', numpy.asarray((unique,
counts)).transpose())

sample3 = sample_wr(x)
unique, counts = numpy.unique(sample3, return_counts = True)
print('Sample 3:\n', numpy.asarray((unique,
counts)).transpose())

sample4 = sample_wr(x)
unique, counts = numpy.unique(sample4, return_counts = True)
print('Sample 4:\n', numpy.asarray((unique,
counts)).transpose())
```

Week 13 Create Bootstrap Sample.py

Bootstrap Method

[data value, frequency count]

Original Sample:

```
[[ 1   1]
 [ 2   1]
 [ 3   1]
 [ 4   1]
 [ 5   1]
 [ 6   1]
 [ 7   1]
 [ 8   1]
 [10   1]]
```

Sample 1:

```
[[ 1.   1.]
 [ 2.   2.]
 [ 3.   1.]
 [ 4.   1.]
 [ 7.   1.]
 [ 8.   1.]
 [10.   2.]]
```

Sample 2:

```
[[ 1.   2.]
 [ 3.   1.]
 [ 4.   1.]
 [ 5.   2.]
 [ 6.   1.]
 [10.   2.]]
```

Sample 3:

```
[[1. 1.]
 [2. 1.]
 [3. 1.]
 [4. 2.]
 [5. 3.]
 [6. 1.]]
```

Sample 4:

```
[[ 1.   1.]
 [ 2.   2.]
 [ 3.   2.]
 [ 5.   2.]
 [ 6.   1.]
 [10.   1.]]
```

Bootstrap Method

Bootstrapping Using Python:

1. In each bootstrap iteration:
   a) Create a bootstrap sample
   b) Calculate the statistic of interest accordingly
   c) Append the statistic of interest to a dataset
2. Derive the summary on the statistic of interest
   a) Generate the empirical distribution
   b) Find a pair of values that enclose 95% of values
   c) Calculate the standard deviation of values

Bootstrap Method: Mean Examples

Common Specifications

- Input Data
  - Number of observations is 100
  - Data are simulated from a particular distribution
  - Random seed is 20191113
  - The summary statistic is the Mean
- Bootstrapping
  - Get the standard error and 95% confidence interval of the summary statistic
  - Perform 500 and 1000 times

```
random.seed(a = 20191113)
population = numpy.zeros((nPopulation,1))
for i in range(nPopulation):
    population[i] = random.gauss(10,7)
zConfInterval (population)
```

Week 13 Bootstrap Mean Example.py

Bootstrap Method: Mean Examples

```
# Calculate the means for the bootstrap samples
def bootstrap_mean (inData, nB):
   n = len(inData)
   outMean = numpy.empty((nB,1))
   for iB in range(nB):
      bootSample = sample_wr(inData)
      bootMean = 0
      for j in range(n):
          bootMean = bootMean + bootSample[j]
      outMean[iB] = bootMean / n
   return outMean
```

Week 13 Bootstrap Mean Example.py

```
# Calculate the 95% confidence interval based on the normal distribution
def zConfInterval (inData):
    n = len(inData)
    obsMean = inData.mean()
    obsSE = inData.std() / numpy.sqrt(n)
    z = 1.9599639845
    lowerCI = obsMean - z * obsSE
    upperCI = obsMean + z * obsSE
    print('Observed Mean: {:.7f}' .format(obsMean))
    print('Observed Standard Error: {:.7f}' .format(obsSE))
    print('95% z-confidence interval {:.7f}, {:.7f}:' .format(lowerCI, upperCI))
```

Bootstrap Method: Mean Examples

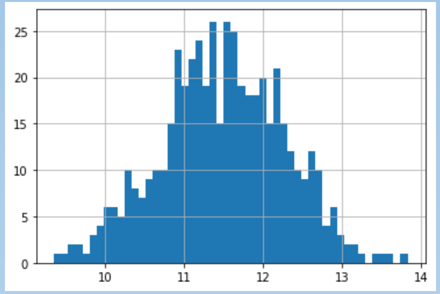
Summarize the bootstrap results to obtain estimates for the statistic's standard error, 95% confidence interval, and sample distribution

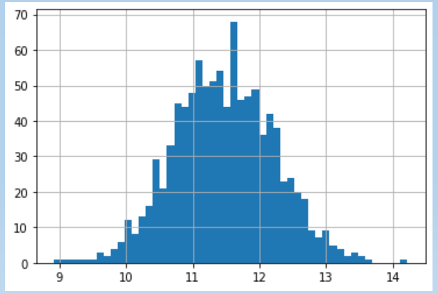
```
# Summarize the bootstrap results
def summarize_bootstrap (bootResult):
    print('Bootstrap Statistics:\n')
    print('                Number:', len(bootResult))
    print('                  Mean: {:.7f}' .format(bootResult.mean()))
    print('    Standard Deviation: {:.7f}' .format(bootResult.std()))
    print('95% Confidence Interval: {:.7f}, {:.7f}'
          .format(numpy.percentile(bootResult, (2.5)), numpy.percentile(bootResult, (97.5))))

    plt.hist(bootResult, align = 'mid', bins = 50)
    plt.grid(axis='both')
    plt.show()
```

Week 13 Bootstrap Mean Example.py

Bootstrap Method: Mean Example 1

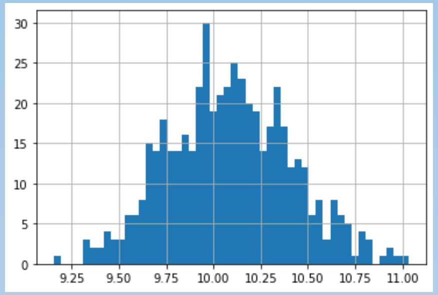
- Input Data
  - Simulated Distribution: Normal, mean = 10, standard deviation = 7
  - Observed Mean (⍰ z-Standard Error) is 11.4698122 (⍰ 0.7502696)
  - 95% z-Confidence Interval is (9.9993108, 12.9403136)
- Bootstrapping 500 Times of the Mean Statistic
  - Mean is 11.4909977 and Standard Deviation is 0.7785430
  - 95% Confidence Interval is (9.9347357, 12.9175503)
- Bootstrapping 1000 Times of the Mean Statistic
  - Mean is 11.4699720 and Standard Deviation is 0.7517242
  - 95% Confidence Interval is (10.0187111, 12.9595051)

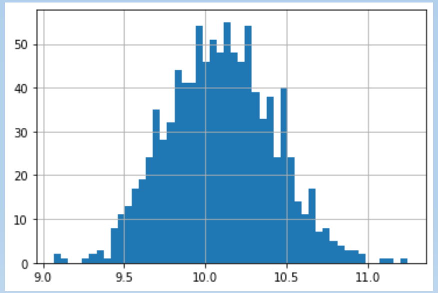




Bootstrap Method: Mean Example 2

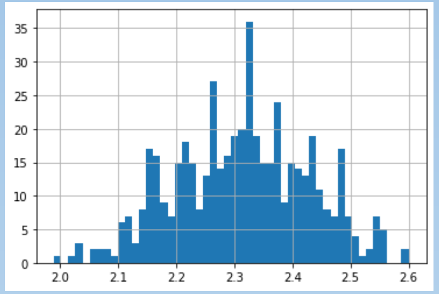
- Input Data
  - Simulated Distribution: Gamma, mean = 10.5, SD = $\sqrt{10.5}$ = 3.2403703
  - Observed Mean (⍰ z-Standard Error) is 10.0751304 (⍰ 0.3229253)
  - 95% z-Confidence Interval is (9.4422085, 10.7080523)
- Bootstrapping 500 Times of the Mean Statistic
  - Mean is 10.0871907 and Standard Deviation is 0.3255313
  - 95% Confidence Interval is (9.4818758, 10.7433969)
- Bootstrapping 1000 Times of the Mean Statistic
  - Mean is 10.0954962 and Standard Deviation is 0.3273110
  - 95% Confidence Interval is (9.4988971, 10.7348476)

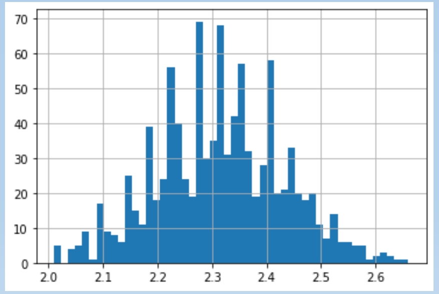




Bootstrap Method: Mean Example 3

- Input Data
  - Simulated Distribution: Pr(X=1)=0.3, Pr(X=2)=0.1, Pr(X=3)=0.4, Pr(X=4)=0.2
  - E(X) = 2.5, Var(X) = 1.25, SD(X) = 1.1180340
  - Observed Mean (⍰ z-Standard Error) is 2.3100000 (⍰ 0.1172135)
  - 95% z-Confidence Interval is (2.0802658, 2.5397342)
- Bootstrapping 500 Times of the Mean Statistic
  - Mean is 2.3136800 and Standard Deviation is 0.1157310
  - 95% Confidence Interval is (2.1000000, 2.5400000)
- Bootstrapping 1000 Times of the Mean Statistic
  - Mean is 2.3146500 and Standard Deviation is 0.1168857
  - 95% Confidence Interval is (2.0900000, 2.5500000)





Bootstrap Method: CV Examples

Common Specifications

- Input Data
  - Number of observations is 100
  - Data are simulated from a particular distribution
  - Random seed is 20191113
  - The summary statistic is the Coefficient of Variation = SD / Mean (Note: this is a biased estimator of the population CV)
- Bootstrapping
  - Get the standard error and 95% confidence interval of the summary statistic
  - 500 and 1000 times

Week 14 Bootstrap CV Example.py

Bootstrap Method: CV Examples

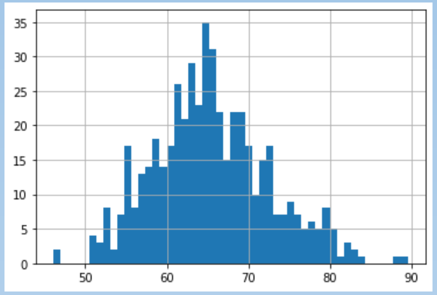
Standard Error of the Coefficient of Variation

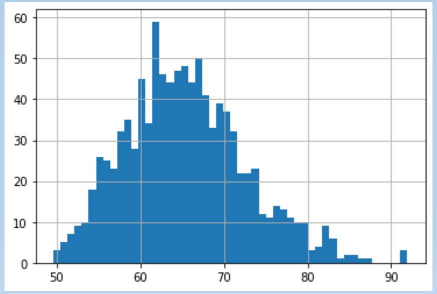
- This website http://influentialpoints.com/Training/standard_error_of_coefficient_of_variation.htm gives a formula for the asymptotic standard error for normal distribution data
- $\mathrm{se}(CV) = \frac{CV}{\sqrt{2n}}\sqrt{1 + 2\left(\frac{CV}{100}\right)^2}$ where CV is expressed in percentages
- We can check this asymptotic standard error against the bootstrap results.

Week 14 Bootstrap CV Example.py

Bootstrap Method: CV Example 1

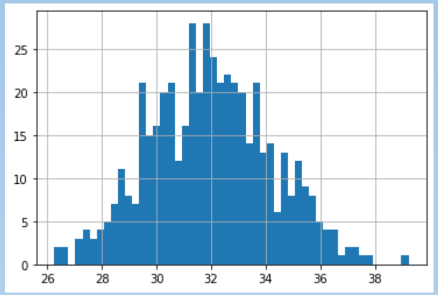
- Input Data
  - Simulated Distribution: Normal, mean = 10, standard deviation = 7, CV = 70%
  - Observed CV (⍰ z-Standard Error) is 65.4125456% (⍰ 6.3009638%)
  - 95% z-Confidence Interval is (53.0628835%, 77.7622077%)
- Bootstrapping 500 Times of the CV Statistic
  - Mean is 65.3862739% and Standard Deviation is 7.0765185%
  - 95% Confidence Interval is (52.5474446%, 80.3672691%)
- Bootstrapping 1000 Times of the CV Statistic
  - Mean is 65.4251694% and Standard Deviation is 7.2087710%
  - 95% Confidence Interval is (53.1530442%, 81.6593212%)

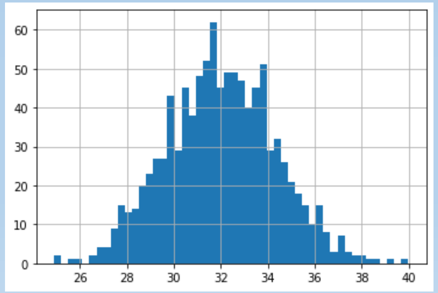




Bootstrap Method: CV Example 2

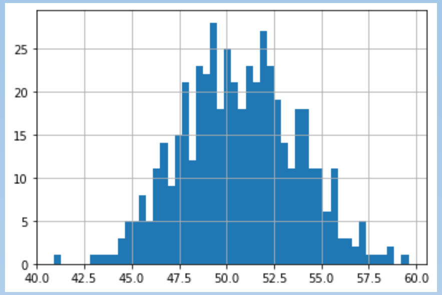
- Input Data
  - Simulated Distribution: Gamma, mean = 10.5, CV =$1/\sqrt{10.5}$ = 30.86%
  - Observed CV (⍰ z-Standard Error) is 32.0517209% (⍰ 2.4883600%)
  - 95% z-Confidence Interval is (27.1746248%, 36.9288169%)
- Bootstrapping 500 Times of the CV Statistic
  - Mean is 31.9818172% and Standard Deviation is 2.2344820%
  - 95% Confidence Interval is (27.7704495%, 36.3145352%)
- Bootstrapping 1000 Times of the CV Statistic
  - Mean is 32.0214920% and Standard Deviation is 2.3023207%
  - 95% Confidence Interval is (27.6705634%, 36.4458927%)

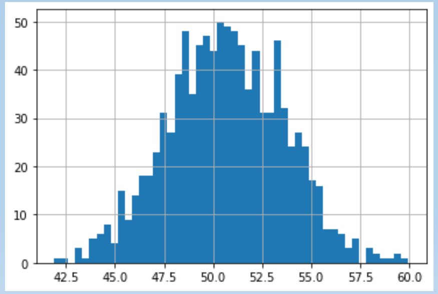




Bootstrap Method: CV Example 3

- Input Data
  - Simulated Distribution: Pr(X=1)=0.3, Pr(X=2)=0.1, Pr(X=3)=0.4, Pr(X=4)=0.2
  - E(X) = 2.5, Var(X) = 1.25, theoretical CV is 44.72%
  - Observed CV (± z-Standard Error) is 50.7417664% (± 4.4162034%)
  - 95% z-Confidence Interval is (42.0861668%, 59.3973661%)
- Bootstrapping 500 Times of the CV Statistic
  - Mean is 50.6918017% and Standard Deviation is 3.0381406%
  - 95% Confidence Interval is (44.9900869%, 56.5142718%)
- Bootstrapping 1000 Times of the CV Statistic
  - Mean is 50.6632044% and Standard Deviation is 2.9507101%
  - 95% Confidence Interval is (44.7786547%, 56.2430608%)





Bootstrap Method

Closing Remarks:

- The Confidence Interval is simple to use, but actual probability coverage may be less than the nominal value, e.g., 95%.
- The bootstrap method depends on the sample being representative. Otherwise, the re-sampling results will be inappropriate.
- If the sample statistic of interest (e.g., the Coefficient of Variation) is <u>already a biased estimate</u> of the population statistic, then the bootstrap method cannot correct the problem though the bootstrap standard error and confidence interval are accurate

Ensemble Learners

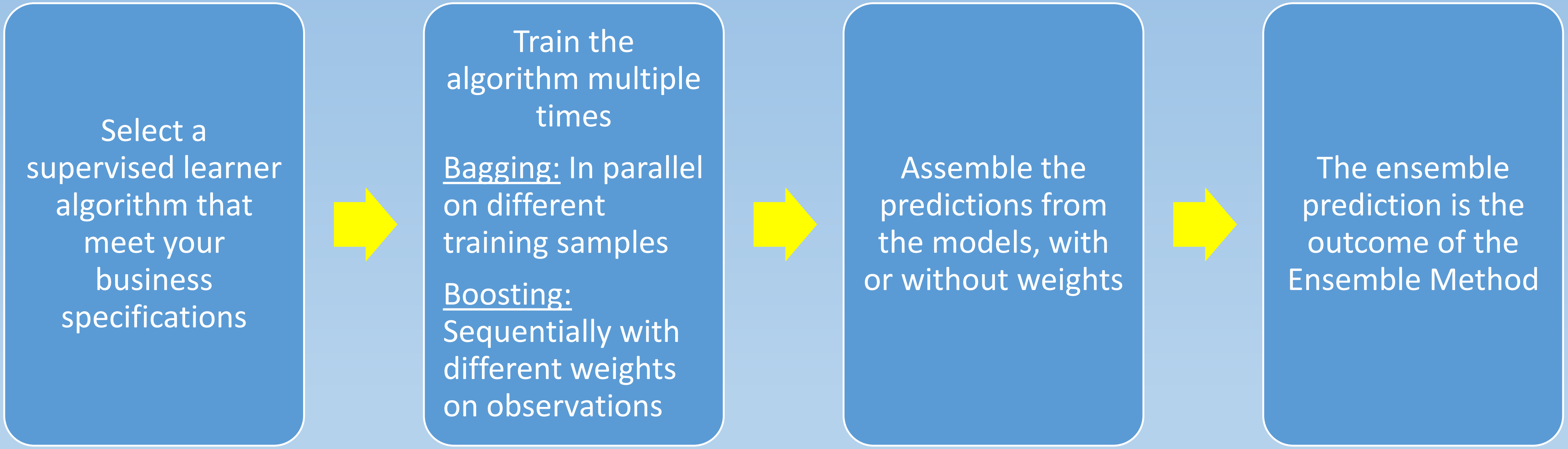
The idea is to train a series of models using the same learning algorithm.

The objective is to combine the models and create a stronger model that delivers better performance.

Bagging averages a series of model outcomes which have high variance to decrease the outcome variance.

Boosting builds a series of incremental models to decrease the bias, while keeping variance small.

Bagging or Boosting: Overview



Select a supervised learner algorithm that meet your business specifications

Train the algorithm multiple times

Bagging: In parallel on different training samples

Boosting: Sequentially with different weights on observations

Assemble the predictions from the models, with or without weights

The ensemble prediction is the outcome of the Ensemble Method

Bagging

Bagging is Bootstrap Aggregating:

1. Create a bootstrap sample from the training partition
2. The model is trained individually on each bootstrap sample
3. Observations in a bootstrap sample are not weighted
4. Models’ predicted target values are summarized (without weights) to produce the ensemble prediction
   a) Mean of the predicted values for an interval target
   b) Mean of the predicted probabilities for a category of a categorical target
   c) Mode of the predicted outcome of a categorical target (majority vote)

Bagging Example: HMEQ

- Interval predictors: DELINQ and DEBTINC
- Binary target: BAD (1 = Event, 0 = Non-Event)
- 5960 observations read
  - 4,217 observations used after removing missing values in the predictors and the target variable
- Assign the observations into the Training and the Testing partitions, stratified by the target variable
  - 70% for Training, actually has 2,951 observations
  - 30% for Testing, actually has 1,266 observations

Week 13 Bagging ClassTree Example.py

Bagging Example: HMEQ

- Classification of Observations
  - Classify an observation as an Event if the predicted probability is greater than or equal to a threshold value
  - The threshold is the fraction of BAD = 1 observations in the Training partition
  - The threshold value is 0.09115554049474754

- Classification Tree Specifications
  - Criterion is Entropy
  - Maximum depth is two (i.e., the tree has three layers including the root node)
  - Random seed is 60616

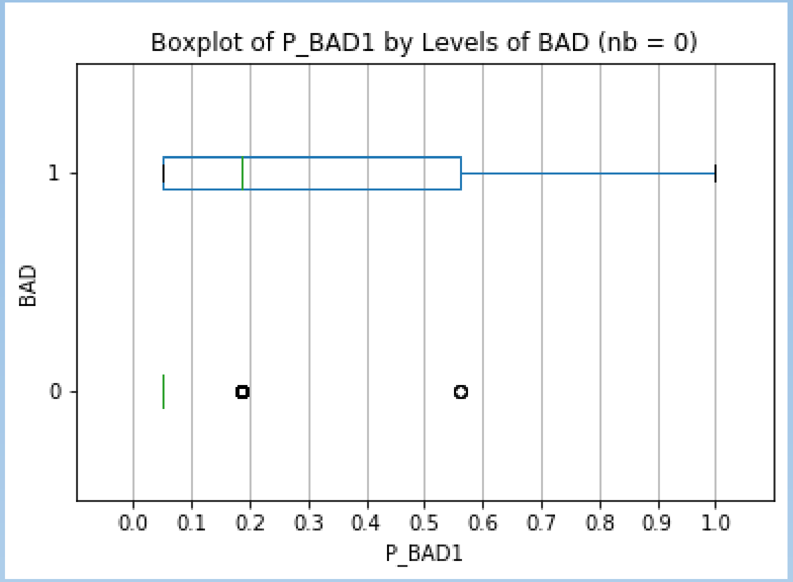
Bagging Example: HMEQ

```
hmeq = pandas.read_csv('C:\\IIT\\Machine Learning\\Data\\hmeq.csv',
                       usecols = ['DELINQ', 'DEBTINC', 'BAD'])
hmeq = hmeq.dropna()

# Partition the data, 70% for training and 30% for testing
x_train, x_test, y_train, y_test = model_selection.train_test_split(hmeq[['DELINQ', 'DEBTINC’]],
   hmeq[['BAD']], test_size = 0.3, random_state = 20191113, stratify = hmeq[['BAD']])

# Calculate the threshold value for declaring a predicted event
threshold = len(y_train[y_train['BAD'] == 1]) / len(y_train)
print('Threshold Value for Declaring a Predicted Event = {:.7f}' .format(threshold))

# Build a classification tree on the training partition
classTree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=60616)
treeFit = classTree.fit(x_train, y_train['BAD'])
treePredProb = classTree.predict_proba(x_test)
AUC_test, RASE_test, MisClassRate_test = ModelMetrics (y_test['BAD'].values, 1, treePredProb[:,1],
                                                       threshold)
```

Bagging Example: HMEQ

Classification Tree Without Bagging:

- Metrics are calculated on the Testing partition
- Area Under Curve = 0.7097004
- Misclassification Rate = 0.1895735
- Root Average Squared Error = 0.2500173



Comments

- The classification tree result is fairly good
- Can we improve further?

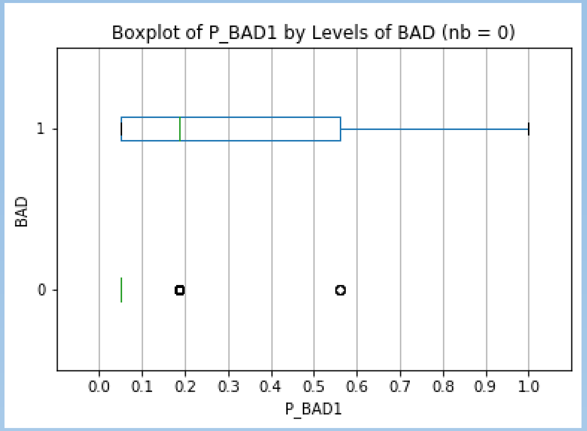
Bagging Example: HMEQ

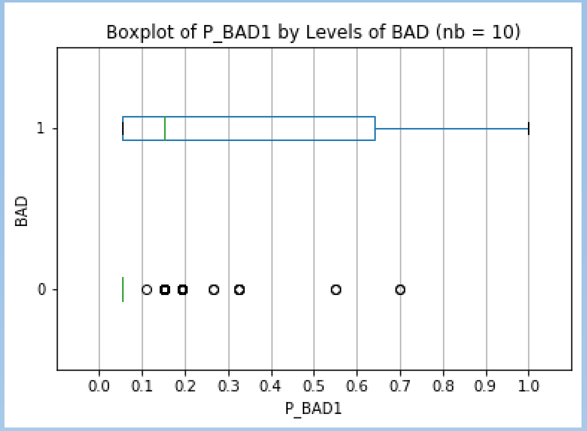
Classification Tree With Bagging:

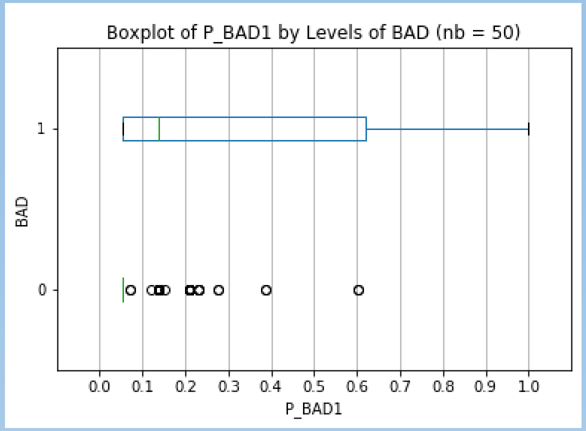
```
def bootstrap_classTree (x_train, y_train, x_test, nB):
   x_index = x_train.index
   nT = len(x_test)
   outProb = numpy.zeros((nT,2))
   classTree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=60616)
   random.seed(20181114)
   for iB in range(nB):
      bootIndex = sample_wr(x_index)
      x_train_boot = x_train.loc[bootIndex[:,0]]
      y_train_boot = y_train.loc[bootIndex[:,0]]
      treeFit = classTree.fit(x_train_boot, y_train_boot['BAD'])
      outProb = outProb + classTree.predict_proba(x_test)
   outProb = outProb / nB
   return outProb
```

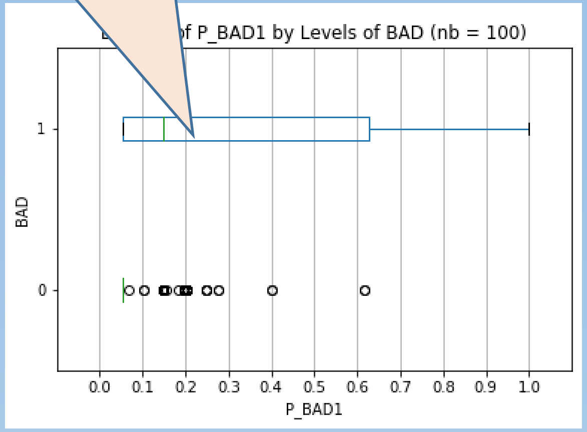
Bagging Example: HMEQ

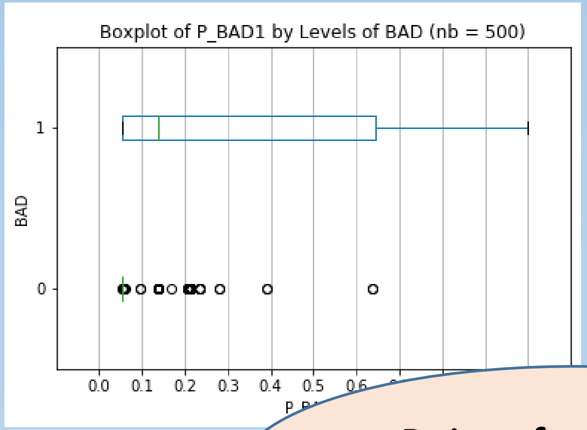
Median for BAD=1 gets smaller

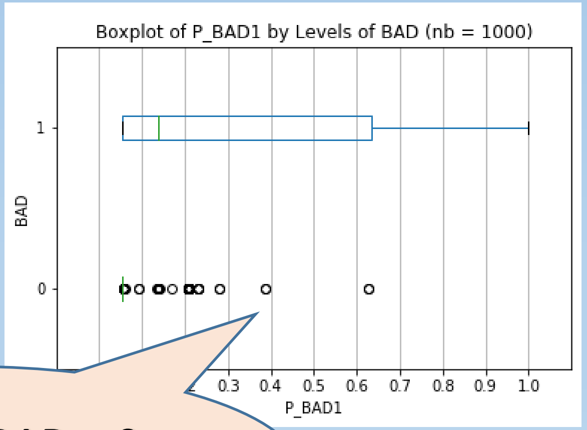


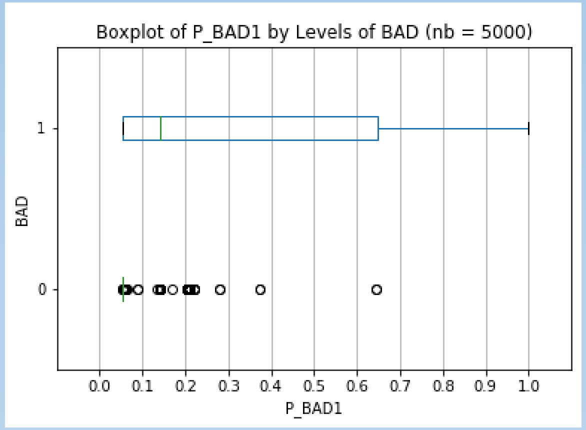


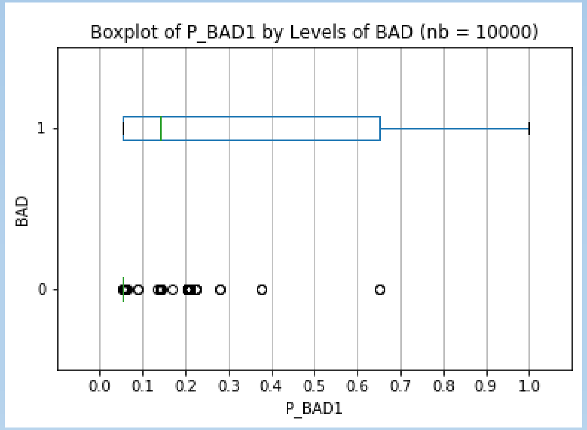












Points for BAD = 0 spread further apart

Bagging Example: HMEQ

Number of Bootstraps	Area Under Curve	Misclassification Rate	Root Average Squared Error
0 (No Bagging)	0.7097004	0.1895735	0.2500173
10	0.7245798	0.1919431	0.2479301
50	0.7238772	0.1919431	0.2465820
100	0.7249537	0.1935229	0.2466624
500	0.7251426	0.1935229	0.2465695
1000	0.7313980	0.1927330	0.2466442
5000	0.7282137	0.1919431	0.2466952
10000	0.7282137	0.1919431	0.2467116

Bagging in Python

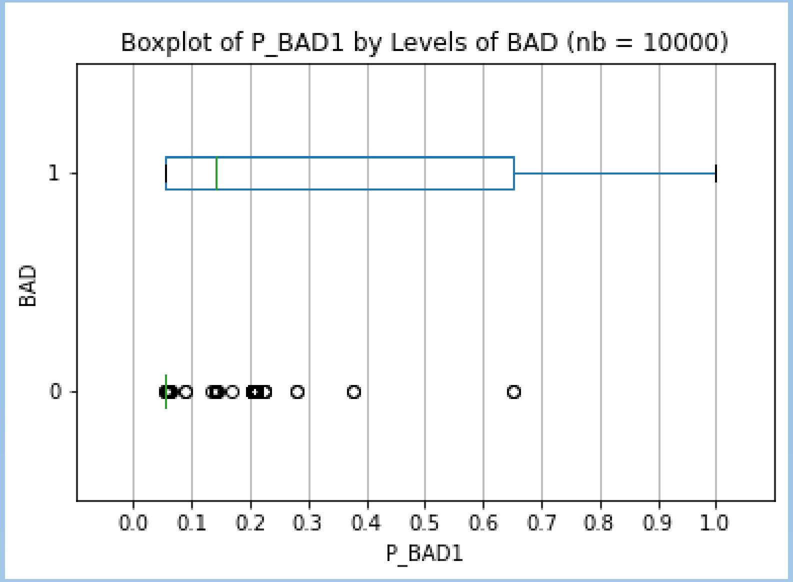
- You can, of course, implement your own Bagging function
- Python offers efficient ways to perform Bagging
  - sklearn.ensemble.BaggingClassifier
  - sklearn.ensemble.BaggingRegressor

Bagging in Python

```
nB = 10000
classTree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=60616)
bagTree = ensemble.BaggingClassifier(base_estimator = classTree, n_estimators = nB, random_state = 20191113,
verbose = 1)
bagFit = bagTree.fit(x_train, y_train['BAD'])
bagPredProb = bagFit.predict_proba(x_test)

AUC_test, RASE_test, MisClassRate_test = ModelMetrics (y_test['BAD'].values, 1, bagPredProb[:,1], threshold)

y_test['P_BAD1'] = treePredProb[:,1]
y_test.boxplot(column='P_BAD1', by='BAD', vert = False, figsize=(6,4))
plt.title("Boxplot of P_BAD1 by Levels of BAD (nb = " + str(nB) + ')')
plt.suptitle(" ")
plt.xlabel("P_BAD1")
plt.ylabel("BAD")
plt.xlim(-0.1,1.1)
plt.grid(axis="y")
plt.xticks(numpy.arange(0.0, 1.1, 0.1))
plt.show()

print('      Number of Bootstraps: ', nB)
print('          Area Under Curve: {:.7f}' .format(AUC_test))
print('    Misclassification Rate: {:.7f}' .format(MisClassRate_test))
print('Root Average Squared Error: {:.7f}' .format(RASE_test))
```

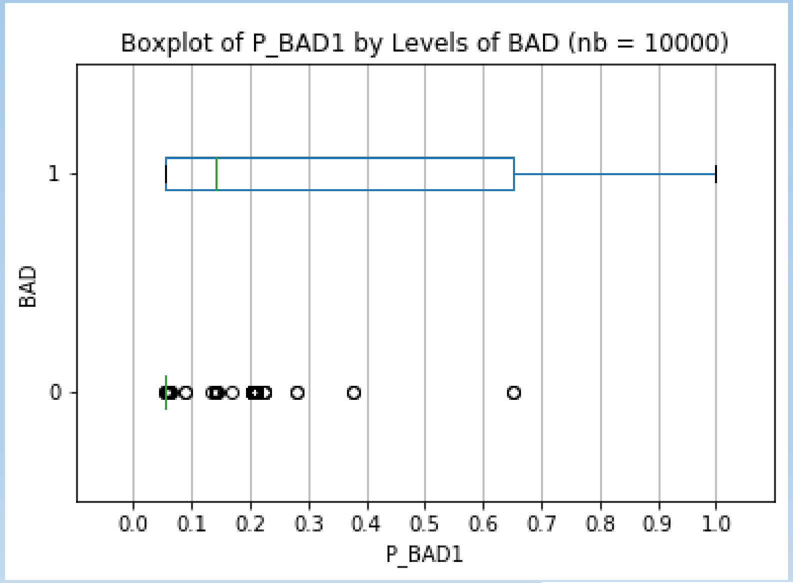
Bagging in Python

ensemble.BaggingClassifier
- Number of Bootstraps: 10000
- Area Under Curve: 0.7228233
- Misclassification Rate: 0.1919431
- Root Average Squared Error: 0.2467374



Faster

Home-grown Bagging
- Number of Bootstraps: 10000
- Area Under Curve: 0.7282137
- Misclassification Rate: 0.1919431
- Root Average Squared Error: 0.2467116

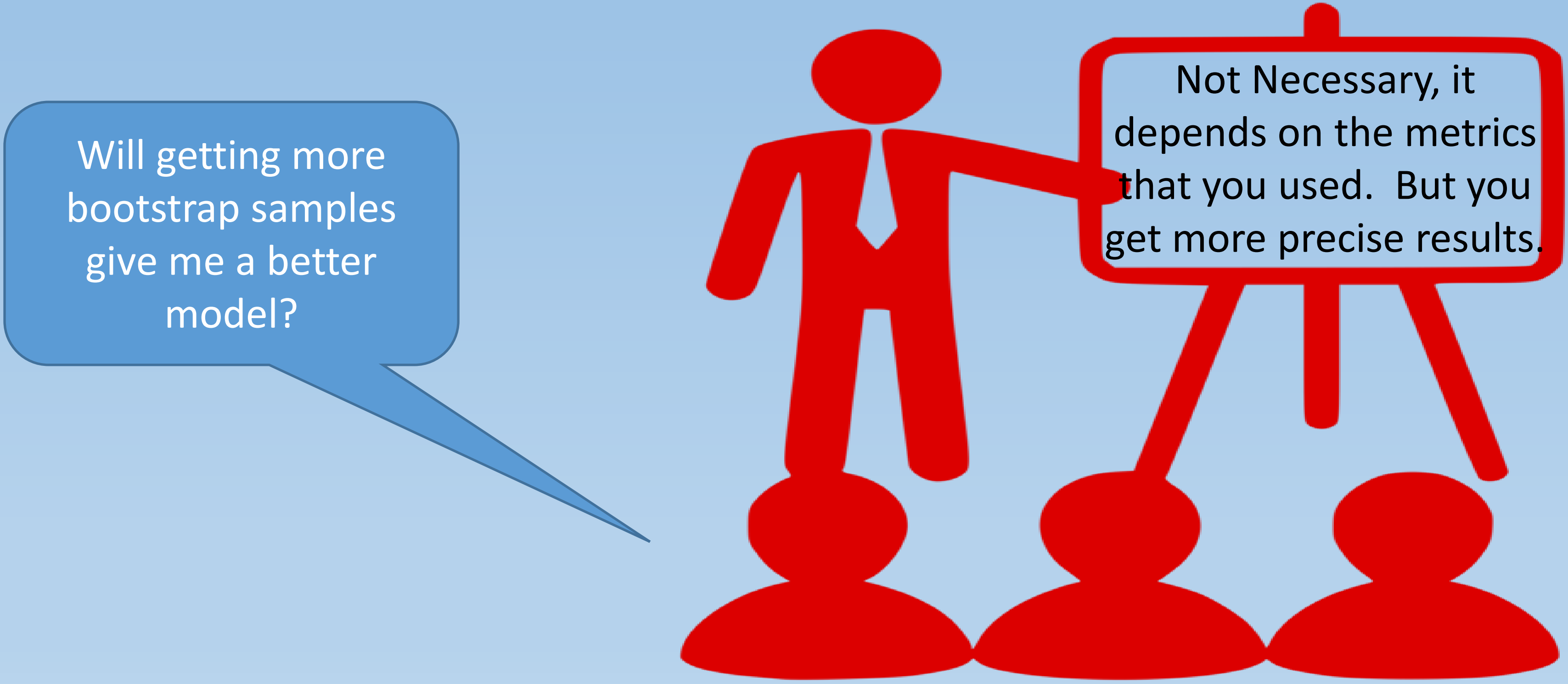


Slower but Flexible

Bagging Example: HMEQ

- We use the Bootstrap method to estimate the distribution of the predicted probabilities Prob(BAD = 0) and Prob(BAD = 1).
- The more bootstrap iterations performed, the distributions will be more precise (a layman term for the word *Converged*).
- Indirectly, the Area Under Curve, the Misclassification Rate, and the Root Average Squared Error metrics will be more precise for large enough number of bootstrap iterations.

Bagging: Number of Bootstrap Samples


Will getting more bootstrap samples give me a better model?
Not Necessary, it depends on the metrics that you used. But you get more precise results.

Bagging: Should I Do It?

- Implementation of Bagging is straightforward
- Bagging can be applied to any supervised predictive models
- Bagging will, in general, improve model goodness-of-fit, but for a price of model interpretation
  - Because the bagging builds the same model on multiple training data
  - Then aggregates the multiple outcomes (e.g., predicted probabilities).
- Bagging will consume more machine computing cycles and resources
  - Specify your number of bootstrap samples based on how long you can wait for the results.

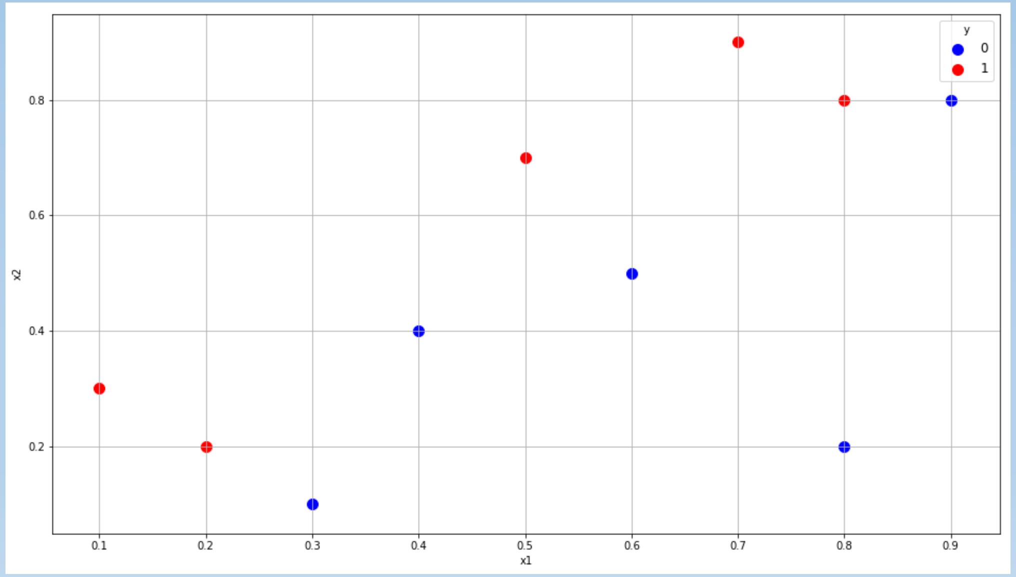
Adaptive Boosting (AdaBoost)

Boosting adaptively weighs observations:

1. The model is trained on the training partition (without weights)
2. Poorly predicted observations are assigned larger weights, such that subsequent model will try to improve their prediction accuracies
3. Iterate Step 2 until the overall model performance is desirable
4. Outcomes from the models (e.g., predicted probabilities) are aggregated with weights to produce the ensemble prediction
    - A model with poor model performance is given less weight
    - A model with good model performance is given more weight

Boosting: Toy Example

- Ten observations
- Two interval predictors X1 and X2
- One binary target Y
  (1 = Event, 0 = Non-Event)

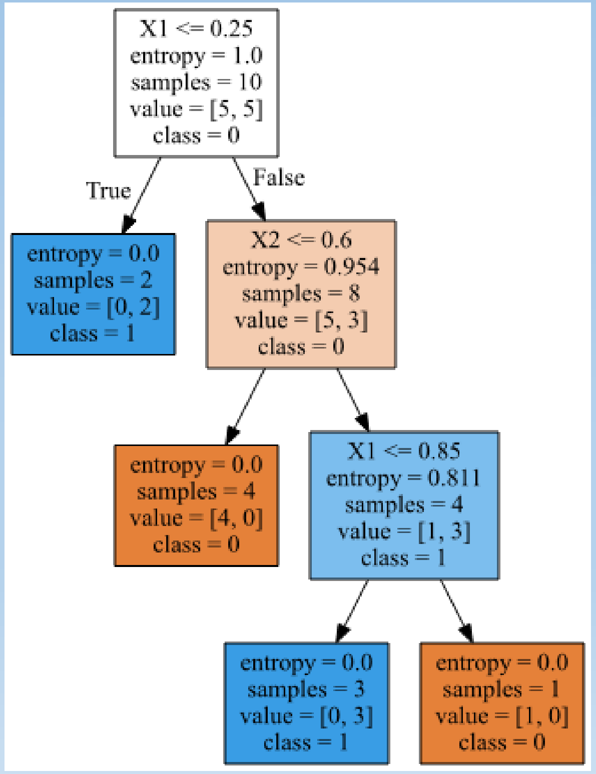


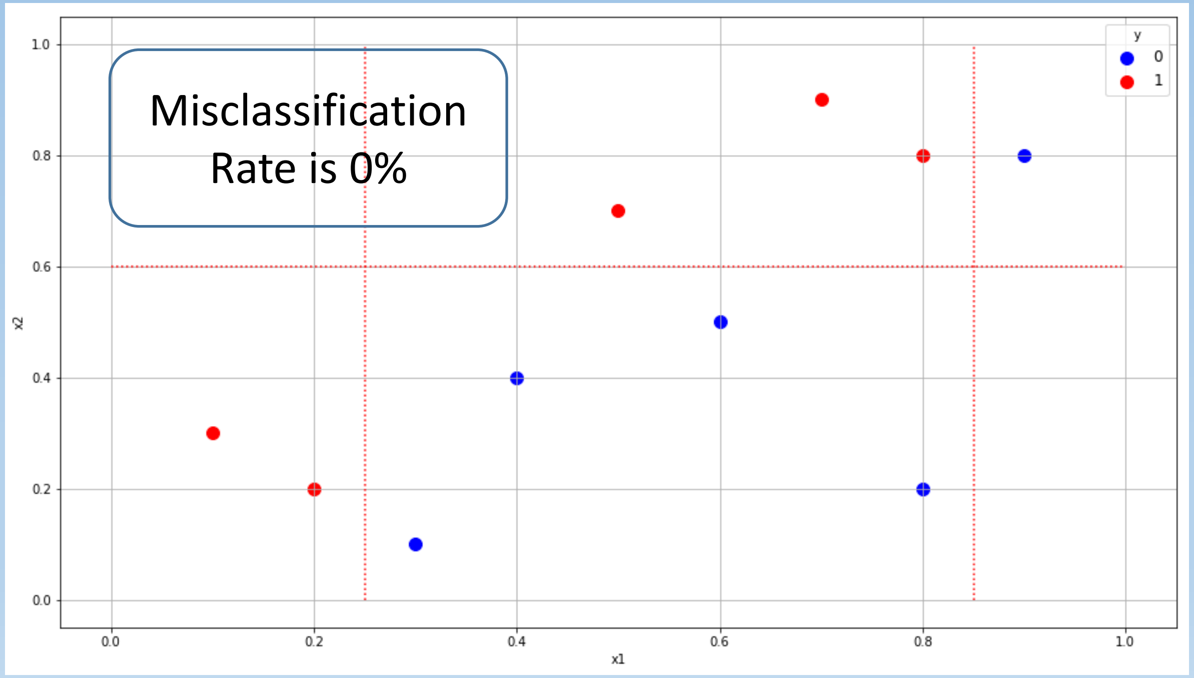
Index	x1	x2	y
0	0.1	0.3	1
1	0.2	0.2	1
2	0.3	0.1	0
3	0.4	0.4	0
4	0.5	0.7	1
5	0.6	0.5	0
6	0.7	0.9	1
7	0.8	0.8	1
8	0.8	0.2	0
9	0.9	0.8	0

Week 13 Toy AdaBoost Example.py

Boosting: A Strong Learner Toy Example

```
classTree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=10, random_state=60616)
treeFit = classTree.fit(x_train, y_train)
treePredProb = classTree.predict_proba(x_train)
accuracy = classTree.score(x_train, y_train)
```





Boosting: A Weak Learner Toy Example

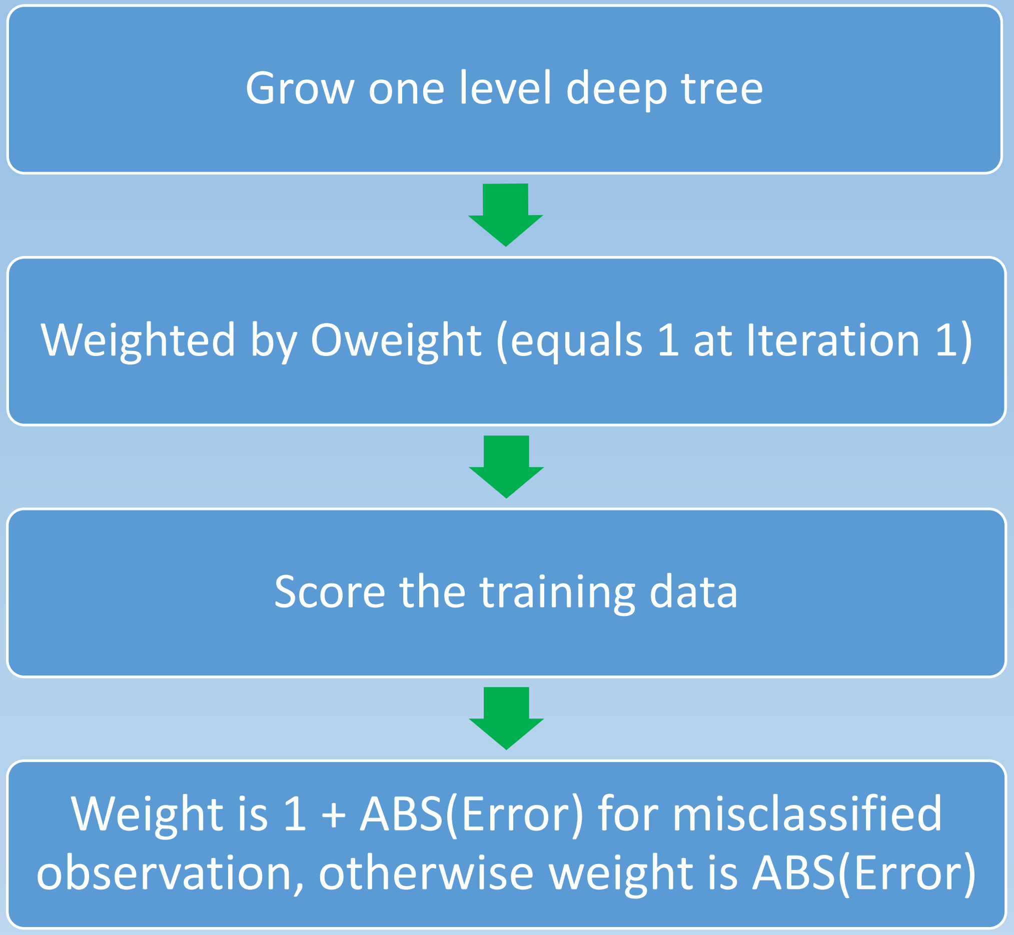
```
x_train = numpy.array([[0.1, 0.3],
                       [0.2, 0.2],
                       [0.3, 0.1],
                       [0.4, 0.4],
                       [0.5, 0.7],
                       [0.6, 0.5],
                       [0.7, 0.9],
                       [0.8, 0.8],
                       [0.8, 0.2],
                       [0.9, 0.8]], dtype = float)

y_train = numpy.array([1, 1, 0, 0, 1, 0, 1, 1, 0, 0], dtype = float)
w_train = numpy.array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype = float)
classTree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=1, random_state=60616)
treeFit = classTree.fit(x_train, y_train, w_train)
treePredProb = classTree.predict_proba(x_train)
AUC = metrics.roc_auc_score(y_train, treePredProb[:,1])
print('Area Under Curve = ', AUC)
```

Suppose the maximum number of depths is restricted to 1

Boosting: Iteration 1

Iteration 1:

```
eventError = numpy.empty((10, 1))
predClass = numpy.empty((10, 1))

for i in range(10):
   if (y_train[i] == 0):
      eventError[i] = 0 - treePredProb[i,1]
   else:
      eventError[i] = 1 - treePredProb[i,1]

   if (treePredProb[i,1] >= treePredProb[i,0]):
      predClass[i] = 1
   else:
      predClass[i] = 0

   if (predClass[i] != y_train[i]):
      w_train[i] = 1 + numpy.abs(eventError[i])
   else:
      w_train[i] = numpy.abs(eventError[i])
```


Grow one level deep tree
Weighted by Oweight (equals 1 at Iteration 1)
Score the training data
Weight is 1 + ABS(Error) for misclassified observation, otherwise weight is ABS(Error)

Boosting: Iteration 1

- Accuracy = 1 – (3 / 10) = 0.7

CaseID	y	Old Weight	Pr(y=0)	Pr(y=1)	Error	Pred. Class	New Weight
1	1	1	0	1	0	1	0
2	1	1	0	1	0	1	0
3	0	1	0.625	0.375	-0.375	0	0.375
4	0	1	0.625	0.375	-0.375	0	0.375
5	1	1	0.625	0.375	0.625	0	1.625
6	0	1	0.625	0.375	-0.375	0	0.375
7	1	1	0.625	0.375	0.625	0	1.625
8	1	1	0.625	0.375	0.625	0	1.625
9	0	1	0.625	0.375	-0.375	0	0.375
10	0	1	0.625	0.375	-0.375	0	0.375

Boosting: Iteration 2

- Accuracy = 1 – (0.375/6.75) = 0.9444444

CaseID	y	Old Weight	Pr(y=0)	Pr(y=1)	Error	Pred. Class	New Weight
1	1	0	1	0	1	0	2
2	1	0	1	0	1	0	2
3	0	0.375	1	0	0	0	0
4	0	0.375	1	0	0	0	0
5	1	1.625	0.0714286	0.928571	0.0714286	1	0.07142857
6	0	0.375	1	0	0	0	0
7	1	1.625	0.0714286	0.928571	0.0714286	1	0.07142857
8	1	1.625	0.0714286	0.928571	0.0714286	1	0.07142857
9	0	0.375	1	0	0	0	0
10	0	*0.375*	0.0714286	0.928571	-0.928571	1	1.92857143
Total		6.75					

Boosting: Iteration 3

- Accuracy = 1 – (0 / 6.1428568) = 1

CaseID	y	Old Weight	Pr(y=0)	Pr(y=1)	Error	Pred. Class	OWeight
1	1	2	0	1	0	1	0
2	1	2	0	1	0	1	0
3	0	0	0	1	-1	1	2
4	0	0	0	1	-1	1	2
5	1	0.07142857	0	1	0	1	0
6	0	0	0	1	-1	1	2
7	1	0.07142857	0	1	0	1	0
8	1	0.07142857	0	1	0	1	0
9	0	0	0	1	-1	1	2
10	0	1.92857143	1	0	0	0	0

Boosting: Toy Example

Iteration	Ensemble Weight (Accuracy)
1	0.7
2	0.944444
3	1.0

Iteration stops since the Weighted Accuracy has reached 1

Ensemble Prediction:

- Boost_P_Y1 =
  (0.7 * P_Y1_Iteration1 + 0.9444444 * P_Y1_Iteration2 + 1 * P_Y1_Iteration3) / (0.7 + 0.944444 + 1)

- Boost_P_Y0 = 1 – Boost_P_Y1

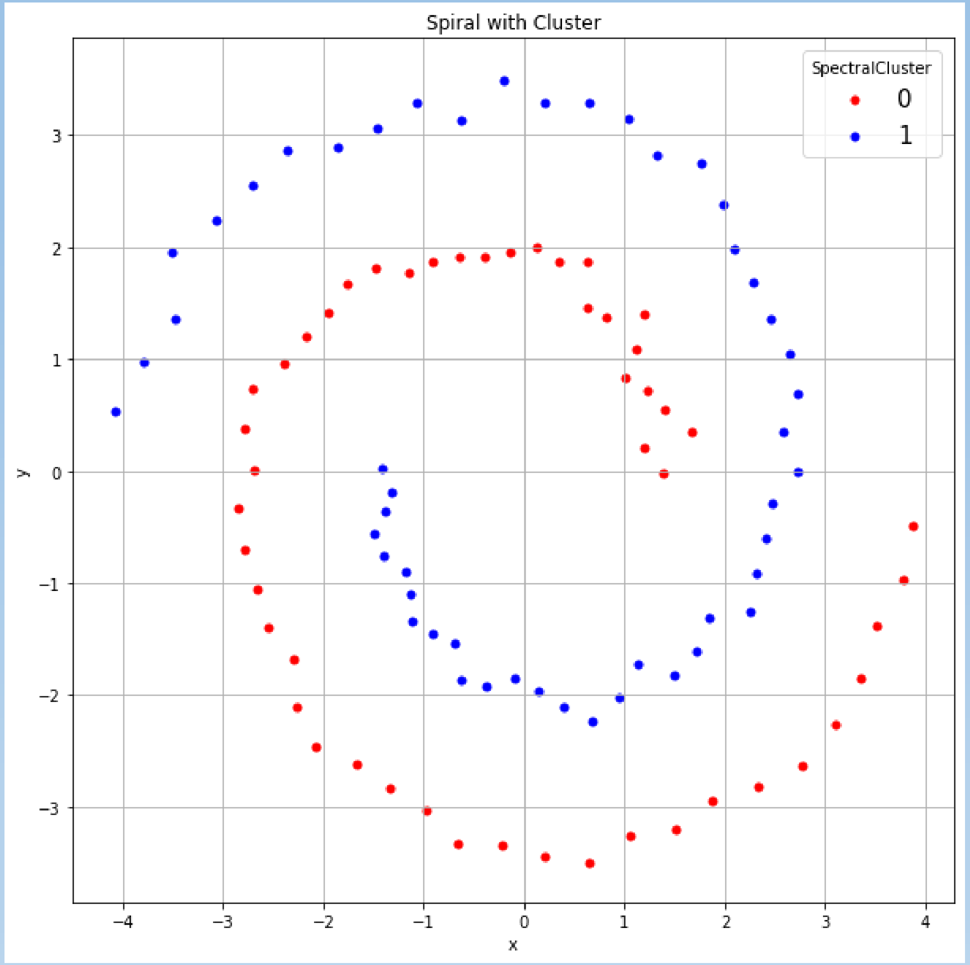
Boosting: Toy Example

caseID	x1	x2	y	Boost_P_Y0	Boost_P_Y1	Boost_Pred_Y	Misclassify?
1	0.1	0.3	1	0.35713	0.64287	1	False
2	0.2	0.2	1	0.35713	0.64287	1	False
3	0.3	0.1	0	0.52258	0.47742	0	False
4	0.4	0.4	0	0.52258	0.47742	0	False
5	0.5	0.7	1	0.19095	0.80905	1	False
6	0.6	0.5	0	0.52258	0.47742	0	False
7	0.7	0.9	1	0.19095	0.80905	1	False
8	0.8	0.8	1	0.19095	0.80905	1	False
9	0.8	0.2	0	0.52258	0.47742	0	False
10	0.9	0.8	0	0.56911	0.43089	0	False

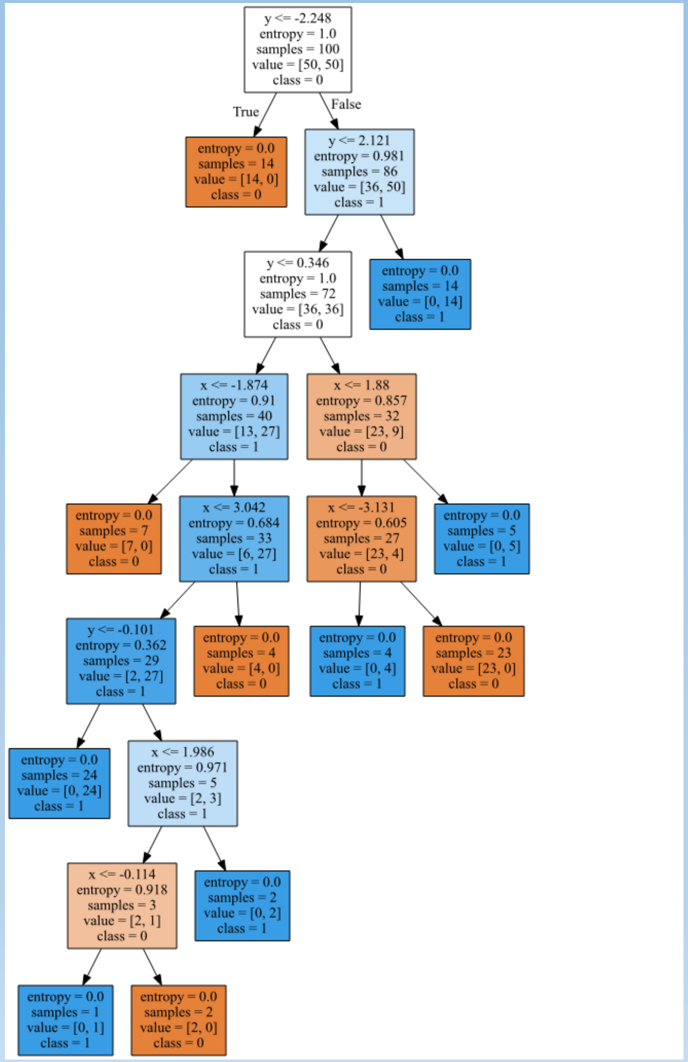
Boosting: Two Spirals

- One hundred observations
- Two interval predictors x and y
- One binary target SpiralCluster (0, 1)

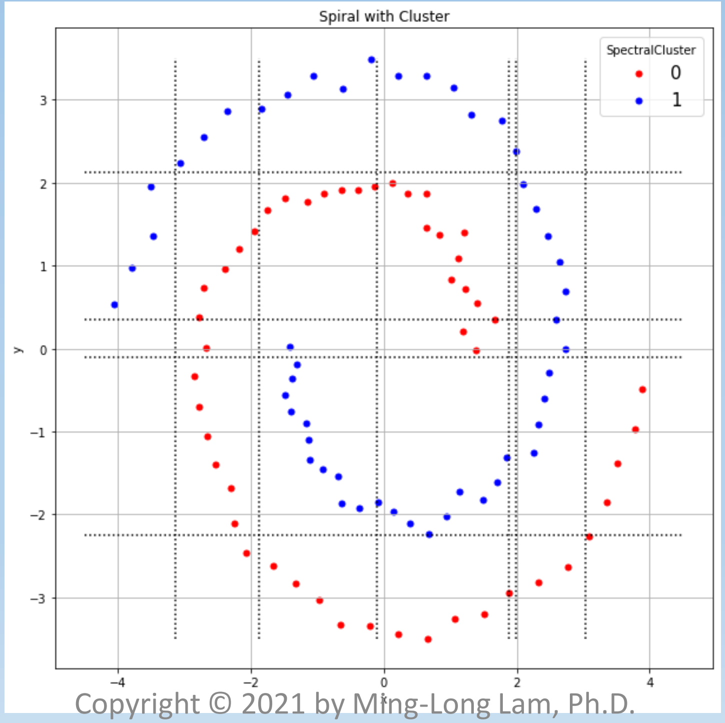
Week 13 Two Spiral AdaBoost.py



Boosting: A Strong Learner Two Spirals



```
classTree = tree.DecisionTreeClassifier(criterion='entropy',
max_depth=10, random_state=60616)
treeFit = classTree.fit(x_train, y_train)
treePredProb = classTree.predict_proba(x_train)
accuracy = classTree.score(x_train, y_train)
```



An eight levels tree gives zero Misclassification Rate

The dotted lines indicate the x-coordinate and the y-coordinate of the split criteria

Boosting: A Weak Learner Two Spirals

```
for iter in range(28):
    classTree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=60616)
    treeFit = classTree.fit(x_train, y_train, w_train)
    treePredProb = classTree.predict_proba(x_train)
    accuracy[iter] = classTree.score(x_train, y_train, w_train)
    ensemblePredProb += accuracy[iter] * treePredProb

    # Update the weights
    eventError = numpy.empty((nObs, 1))
    predClass = numpy.empty((nObs, 1))

    for i in range(nObs):
        if (y_train[i] == 0):
             eventError[i] = 0 - treePredProb[i,1]
        else:
             eventError[i] = 1 - treePredProb[i,1]

        if (treePredProb[i,1] >= treePredProb[i,0]):
           predClass[i] = 1
        else:
           predClass[i] = 0

        if (predClass[i] != y_train[i]):
           w_train[i] = 1 + numpy.abs(eventError[i])
        else:
           w_train[i] = numpy.abs(eventError[i])
```

Suppose the maximum number of depths is restricted to 2

Boosting: A Weak Learner Two Spirals

Iteration	Accuracy
0	0.6400000000000
1	0.8819444444444
2	0.9784898792124
3	0.9874700370451
4	0.9936661745293
5	0.9979558952939
6	0.9987668310317
7	0.9995618142112
8	0.9997583931071
9	0.9999036729736
10	0.9999477679093
11	0.9999780745323
12	0.9999889638162
13	0.9999952586201

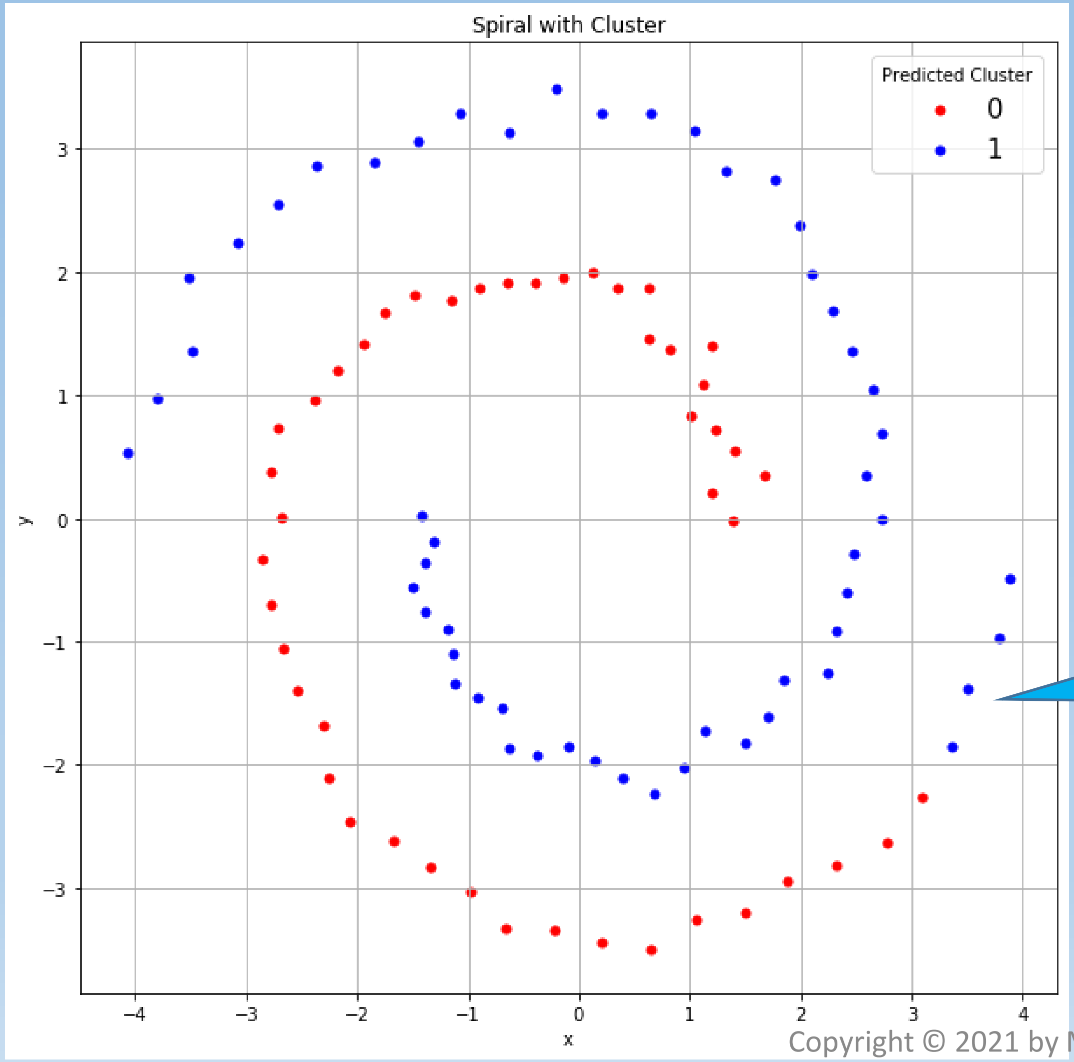
Iteration	Accuracy
14	0.9999976942166
15	0.9999989871230
16	0.9999994968119
17	0.9999997742193
18	0.9999998881804
19	0.9999999544144
20	0.9999999769783
21	0.9999999922207
22	0.9999999961526
23	0.9999999987237
24	0.9999999994206
25	0.9999999998886
26	0.9999999999449
27	1.0000000000000

Attained 100% Accuracy in 28 iterations

Boosting: A Weak Learner Two Spirals

```
# Calculate the final predicted probabilities
ensemblePredProb /= numpy.sum(accuracy)

trainData['predCluster'] = numpy.where(ensemblePredProb[:,1] >= 0.5, 1, 0)


carray = ['red', 'blue']
plt.figure(figsize=(10,10))
for i in range(2):
    subData = trainData[trainData['predCluster'] == i]
    plt.scatter(x = subData['x'],
                y = subData['y'], c = carray[i], label = i, s = 25)
plt.grid(True)
plt.title('Spiral with Cluster')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(title = 'Predicted Cluster', loc = 'best', bbox_to_anchor = (1, 1), fontsize = 15)
plt.show()
```

Boosting: A Weak Learner Two Spirals



Except for these four points, our predictions are almost perfect.

Adaptive Boosting in Python

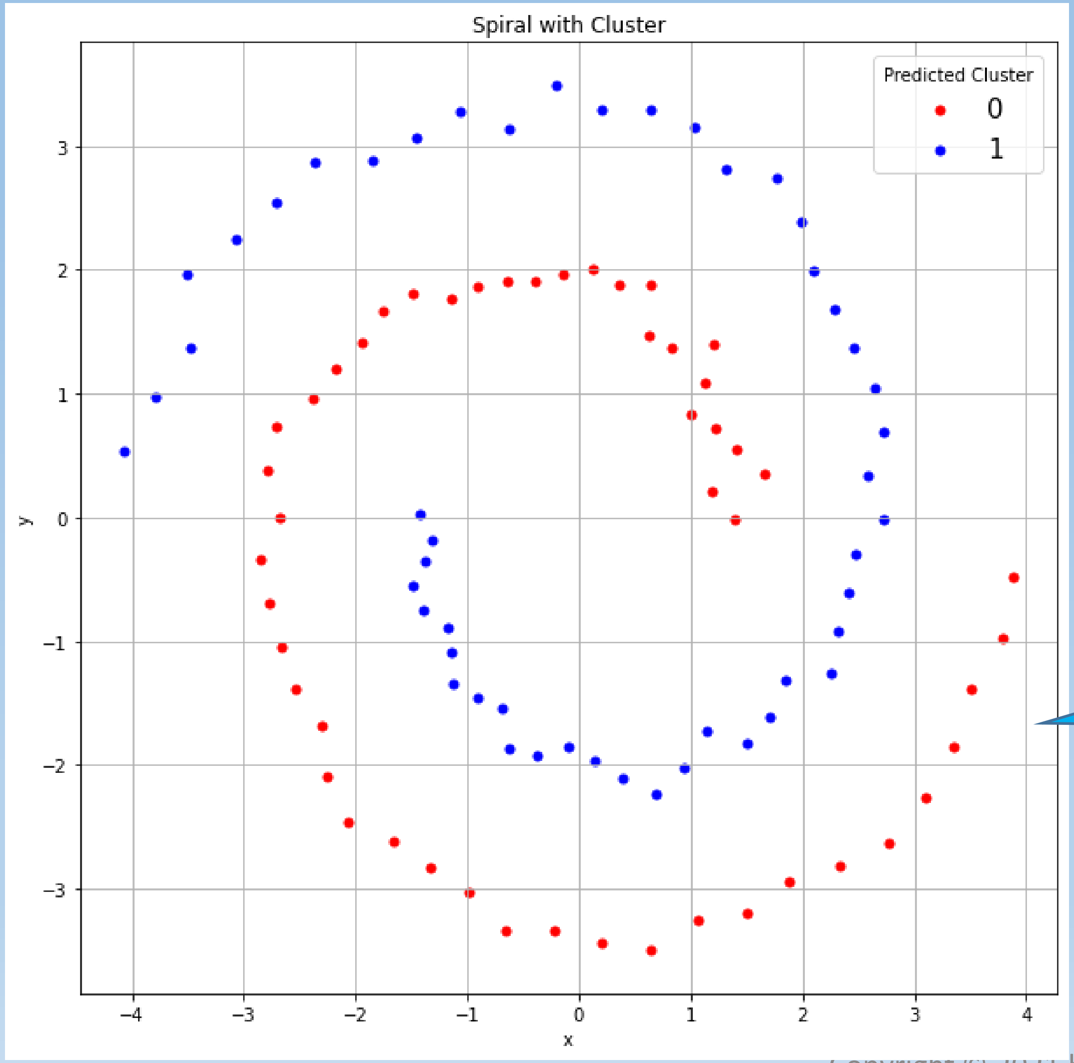
- You can, of course, implement your own Adaptive Boosting function
- Python offers efficient ways to perform Adaptive Boosting
  - sklearn.ensemble.AdaBoostClassifier
  - sklearn.ensemble.AdaBoostRegressor

sklearn.ensemble.AdaBoostClassifier

```
classTree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=60616)
boostTree = ensemble.AdaBoostClassifier(base_estimator=classTree, n_estimators=28,
                                        learning_rate=1.0, algorithm='SAMME.R', random_state=None)
boostFit = boostTree.fit(x_train, y_train)
boostPredProb = boostFit.predict_proba(x_train)
boostAccuracy = boostFit.score(x_train, y_train)

trainData['predCluster'] = numpy.where(boostPredProb[:,1] >= 0.5, 1, 0)
```

Ji Zhu, Hui Zou, Saharon Rosset, and Trevor Hastie (2009) "Multi-class AdaBoost", *Statistics and Its Interface*, Volumne 2, Pages 349-360

sklearn.ensemble.AdaBoostClassifier



Perfect Predictions!
Choice of Weights Matters

Final Remarks

Likes

- Bagging or Boosting often improves model performance
- The algorithm is not difficult to implement

Concerns

- Since multiple models have been trained (parallelly or sequentially), thus we cannot visually represent all the models altogether (e.g., in a single tree diagram)
- The execution time is longer, and special attention needed to avoid a run-away iterative process

Remaining Class Schedule

April 22	Week 14 Class on Gradient Boosting
April 29	Individual 15-minute Q&A meeting via Zoom, Google sheet appointment needed
May 3 to May 6	Take-Home Final Examination, Open on May 3, Deadline on May 6. No Class on May 6

Assignment 5

- Due at 11:59 pm on Sunday, May 2, 2021
- Submit your answers as a PDF file
- Submit your Python codes as .py files, otherwise liable for 50% deduction
- No more than two attempts
- Only grade the most recently submission

Final Examination

- Count 25% toward your final course grade
- Online, open-book, open-note format
- Work offline until you are ready to enter answers online
- Available at 12:01 AM Chicago time on Monday May 3, 2021
- Due at 11:59 PM Chicago time on Thursday May 6, 2021
- Late submission penalty is 2 points per hour past due