HW12 CS 189

1.
Who else did you work with on this homework? In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

I worked on this homework with Ehimare Okoyomon, Prashanth Ganeth, and Daniel Mockaitis. We worked by getting together throughout the week and communicating on facebook.

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up}

Nicholas Lorio, 26089160

Question 5 Own Question.

Q: When is random forest a poor choice for an algorithm?

Accurate ensembles can require more trees, which means using the model becomes slower. In most situations random forest is fast enough, but there are a lot of situations where run-time performance is a necessary constraint and therefore a different algorithm should be used.

Another reason random forest is a poor choice is when we are trying to find the relationships in the data. If it's important to find the relationships in the data then a different algorithm should be used. There is also the problem of overfitting when there are too many high cardinality categorical variables. It is common to run into overfitting with random forests.

https://www.quora.com/When-is-a-random-forest-a-poor-choicerelative-to-other-algorithms

189 - 12

2.a. $\min_w \frac{1}{2}\|y - Xw\|^2 + \lambda\|w\|_1$

$= \min_w \frac{1}{2}\left\| Y - \sum_{i=1}^{d} w_i x_i \right\|^2 + \lambda \sum_{i=1}^{d} |w_i| \quad \text{as } y = \sum_{i=1}^{d} (x_i) w_i$

$= \min_w \frac{1}{2}\left(y^T y + \sum_{i=1}^{d} \left(x_i^T w_i^2 x_i - 2 y^T w_i x_i + \lambda |w_i| \right) \right)$

$= \min \frac{1}{2}\|y\|^2 + \sum_{i=1}^{d} \left(x_i^T w_i^2 x_i - 2 y^T w_i x_i + \lambda |w_i| \right)$

thus $g(y) = \frac{1}{2}\|y\|^2$ & $F(x_i, y, w_i) = x_i^T w_i^2 x_i - 2 y^T w_i x_i + \lambda |w_i|$

2.b. $\hat{w}_i > 0 \quad \hat{w}_i = \min_{w_i} \frac{1}{2}\|y - Xw\|^2 + \lambda \|w\|_1$

~~$= \min_{w_i} \frac{1}{2}\|y\|^2 + \sum_{i=1}^{d} \left(x_i^T w_i^2 x_i - 2 y^T w_i x_i + \lambda |w_i| \right)$~~

- Convex objective function
- use line search method
- Separate terms that depend on w_i

$L(w) = \frac{1}{2}\|Xw - y\|_2^2 + \lambda\|w\|_1 = \frac{1}{2}\left\| \sum_{j=1}^{d} w_i x_j - y \right\|_2^2 + \lambda |w_i| + \lambda \sum_{j \neq i} |w_j|$

let us denote $c^{(1)} = \sum_{j \neq i} w_j x_j - y$ & $c^{(2)} = \lambda \sum_{j \neq i}^{d} |w_j|$

$= \lambda |w_i| + c^{(2)} + \frac{1}{2}\| w_i x_i + c^{(1)} \|_2^2$

$= \lambda |w_i| + c^{(2)} + \frac{1}{2} \sum_{j=1}^{n} \left(w_i x_{j,i} + c_j^{(1)} \right)^2$

$\hat{w}_i > 0$

$$\delta L/\delta w_i = \lambda \cdot 1 + 0 + \frac{1}{2}\sum_{j=1}^{2} (1 \cdot x_{j,i} + 0) \cdot 2(w_i x_{j,i} + c_j^{(1)})^1 = 0$$

$$\lambda + 2\sum x_{j,i}\hat{w}_i^* x_{j,i} + x_{j,i} c_j^{(1)} = 0$$

$$\boxed{w_i^* = \frac{-\lambda + 2\sum_{j=1}^{2} x_{j,i} c_j^{(1)}}{2\sum_{j=1}^{2} x_{j,i}^2}}$$

let $a = 2\sum_{j=1}^{2} x_{j,i} c_j^{(1)}$

$b = 2\sum_{i=1}^{2} x_{j,i}^2$

$$\hat{w}_i^* = \frac{-\lambda + a}{b} \quad \text{for } \hat{w}_i > 0$$

c. $\hat{w}_i < 0$

$$\delta L/\delta w_i = -\lambda + 0 + \sum - x_{j,i} \frac{2}{2}(-w_i x_{j,i} + c_j^{(1)}) = 0$$

$$\hat{w}_i = \frac{\lambda + a}{b} \quad \text{for } \hat{w}_i < 0$$

d. $\hat{w}_i = 0$ if i) $\frac{\lambda + a}{b} \geq 0$ or if ii) $\frac{-\lambda + a}{b} \leq 0$

$\lambda + 2\sum x_{j,i} c_j^{(1)} \geq 0 \quad \lambda \geq -2\sum x_{j,i} c_j^{(1)}$ i) $\lambda \geq -a$

$\lambda \geq 2\sum x_{j,i} c_j^{(1)}$ ii) $\lambda \geq a$

So $\hat{w}_i = 0$ if $\lambda = \pm a$

- if neither $\frac{-\lambda+a}{b} > 0$ or $\frac{\lambda+a}{b} < 0$ then $\nexists$ No optimum

-> lasso is convex so $\exists$ is an opt somewhere, for this case $w_i^* = 0$ thus $\frac{-\lambda+a}{b} \geq 0$ & $\frac{\lambda+a}{b} \leq 0$ which is equivalent to $|a| \leq \lambda$

$(w_i^* = \hat{w}_i)$

i.e. $\hat{w} = \arg\min_w \frac{1}{2}\|y - Xw\|_2^2 + \lambda\|w\|_2^2$

$$\arg\min_{w_i} \frac{1}{2}\Big\|\sum_{j=1}^{d} w_j x_j - y\Big\|_2^2 + \lambda\sum_{j=1}^{d} w_j^2 \qquad \text{convex.}$$

$$\arg\min_{w_i} \frac{1}{2}\Big\|w_i x_i + \underbrace{\sum_{j\neq i} w_j x_j - y}_{C^{(1)}}\Big\|_2^2 + \lambda w_i^2 + \underbrace{\sum_{j\neq i}\lambda w_j^2}_{C^{(2)}}$$

$$\arg\min_{w_i} \frac{1}{2}\sum_{j=1}^{n}\left(w_i x_{ji} + C_j^{(1)}\right)^2 + \lambda w_i^2 + C^{(2)}$$

$$\partial L/\partial w_i = \sum_{j=1}^{n}\frac{2}{2}x_{ji}\left(w_i^* x_{ji} + C_j^{(1)}\right)' + \lambda\cdot 2w_i^* + 0 = 0$$

$$= \frac{2}{2}\sum_{j=1}^{n}\left(x_{ji}w_i^* x_{ji} + x_j C_j^{(1)}\right) + \lambda\cdot 2w_i^* = 0$$

$$= \frac{2}{2}w_i\sum_{j=1}^{n}x_{ji}^2 + \frac{2}{2}\sum_{j=1}^{n}x_{ji}C_j^{(1)} + \lambda\cdot 2w_i^* = 0$$

$$= w_i^*\left(\sum_{j=1}^{n}x_{ji}^2 + 2\lambda\right) + \sum_{j=1}^{n}x_{ji}C_j^{(1)} = 0$$

$$\left[w_i^* = \frac{-\sum_{j=1}^{n}x_{ji}C_j^{(1)}}{\sum_{j=1}^{n}(x_{ji}^2) + 2\lambda} = -a/(b+2\lambda)\right] = \hat{w}_i = 0 \quad \text{when} \quad -\sum_{j=1}^{n}x_{ji}C_j^{(1)} = 0$$

so $\hat{w}_i = 0$ when $a = 0$ or when $\lambda \to \infty$

- l_1 norm promotes sparsity as $\hat{w}_i = 0$ when $\lambda \geq |a|$, by increasing λ we increase the threshold for $|a|$ to be set to zero and our solution becomes more sparse.
- a is some function of the data & other weights.

- the regularization term in l_1 norm tries to pull the least squares update towards zero.

- l_1 norm sparsity for when $|a| \leq \lambda$ we set $w_i := 0$. Because sparse as described previously.

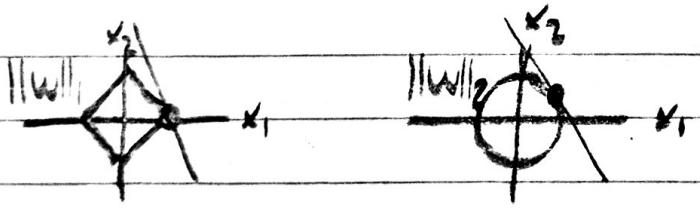
- l_2 norm does not promote sparse solutions. In contrast $\hat{w}_i := 0$ is not dependent on λ in the same manner as l_1 for sparsity.

$\hat{w}_i = 0$ when $\lambda \to \infty$

or

$\hat{w}_i = 0$ when $a = 0$

thus l_1 leads to frequently sparse solutions, many coeff w/ zero values, & l_2 has smaller requirements for $\hat{w}_i = 0$.



l_1 l_2

2.5. (From piazza: "Reporting the hyper-parameter is sufficient". we don't have to include the ipython notebook for this part)

$\lambda = 0.00001$ is a sufficient hyperparameter for l1_reg to extract the image

3.d. $P(\max_i |z_i| \geq 2\sigma(\log(d))^{1/2})$

$= 1 - P(\max_i |z_i| < t) = 1 - P(|z_1| < t, \ldots, |z_d| < t)$

$\geq 1 - \sum_{i=1}^{d} P(|z_i| < t)$ By Hint 2

$\geq 1 - \sum_{i=1}^{d} (1 - P(|z_i| \geq t))$

- $P(|z_i| \geq t) \leq \exp\left(-\frac{(2\sigma(\log(d))^{1/2})^2}{2\sigma^2}\right)$

$\leq \exp\left(-\frac{2^2\sigma^2\log(d)}{2\sigma^2}\right) = \exp(-2\log(d)) = \frac{1}{d^2}$

$\geq 1 - \sum_{i=1}^{d} (1 - P(|z_i| \geq t))$

$\leq 1/d^2$

$\geq 1 - 1/d^2$

$\geq d(1 - 1/d^2)$

$\geq 1 - d(1 - 1/d^2)$

$\geq 1 - d + 1/d \geq 1/d$ as required

as $d \geq 1$ thus $1 - d \leq 0$

3.e. $X \in \mathbb{R}^{n \times d}$ (restricted to orthonormal columns)

Show $\hat{w}_{top}(s)$ returns top s entries of vector $w^* + z'$ in absolute value. • z' iid gauss $N(0, \sigma^2)$

Min $\|y - Xw\|^2$ gives $\hat{w}_{LS}$ for $y = Xw^* + z$

$z = y - Xw$

$\sim N($

$\hat{w}_{top}(s) = T_s(\hat{w}_{LS})$

$X = \begin{bmatrix} | & & | \\ x_1 & \dots & x_d \\ | & & | \end{bmatrix}$, $Xw + z = \begin{bmatrix} | & & | \\ x_1 & \dots & x_d \\ | & & | \end{bmatrix}_{n \times d} \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}_{d \times 1} + \begin{bmatrix} z_1' \\ \vdots \\ z_d' \end{bmatrix}_{d \times 1}$

$\hat{w}_{LS} = X^T y$ • Zero out all but top s entries of $\hat{w}_{LS}$ in absolute value.

as

$= (X^T X)^{-1} X^T y$

$= I^{-1} X^T y$

$= X^T y$

$\hat{w}_{LS} = X^T(Xw^* + z)$ w/ $z \sim$ iid gauss $N(0, \sigma^2)$ aka w/ var σ^2

$= X^T X w^* + X^T z$

$= w^* + X^T z$

• $X^T X = I$

• let $z' = X^T z$

$\hat{w}_{LS} = w^* + z'$

$\begin{bmatrix} z_1' \\ \vdots \\ z_d' \end{bmatrix} = X^T \begin{bmatrix} z_1 \\ \vdots \\ z_d \end{bmatrix}$

• let C be the covariance matrix for z_i.

-> C is a diagonal matrix as z_i are independent.

-> z_i' has a covariance matrix $\hat{C} = X^T C X = \sigma^2 I$

$\hat{C}$ is also a diagonal matrix as X has orthonormal cols & C is a diagonal matrix (as established earlier)

↳ as $cov(Az) = z^T cov(A) z$

3.e. cont.

- z_i are jointly gaussian random variables, each w/s covar matrix given by $\hat{C} = \Psi^T C \Psi$,
  $\Rightarrow \hat{C}$ is diagonal
  thus z_i' are independent R.V.

We can conclude that $\hat{w}_{top(s)}$ returns the s top elms of $w^* + z' = \hat{w}_{ls}$ in abs val w/ z' iid gauss w/ var σ^2.

3.f. $e = \hat{w}_{top(s)} - w^*$ is (@ most) 2s-sparse

$$\hat{w}_{top(s)} = \begin{bmatrix} a+z_i' \\ b+z_i' \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \} s \text{ nonzero}$$

- assume the top s returned from $\hat{w}_{top(s)}$ are the s largest in magnitude of $\hat{w}_{ls}$.

w^* is s-sparse

$$w^* = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \vdots \\ x \\ z \end{bmatrix} \} s \text{ Nonzero}$$

$$e = \begin{bmatrix} a+z_i' \\ b+z_i' \\ \vdots \\ 0 \\ \vdots \\ 0 \\ \vdots \\ -x \\ -z \end{bmatrix} \begin{matrix} \} s \text{ Non zero} \\ \\ \\ \\ \\ \} s \text{ Non zero} \end{matrix}$$

- thus e has @ most 2s non zero & is 2s-sparse (@ most)

3.g. Show $|e_i| \leq 4\sigma(\log(d))^{1/2}$ for each i

$\mathcal{E} = \{e_i\}_{i=1...}$

$\hat{w}_{top}(s) = \begin{bmatrix} b+z_1 \\ b+z_2 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$ s Nonzero $\quad w^* = \begin{bmatrix} b \\ b \\ \vdots \\ b \\ 0 \end{bmatrix}$ s Nonzero

$e_i = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$ s Nonzero z_i

Max error for one is $2\sigma(\log(d))^{1/2}$

thus

$e = (\hat{w}_{top(s)} - w^*) = (\hat{w} + \hat{z}) - w^* = z' \in$

Max $|z_i| \leq 2\sigma\sqrt{\log(d)}$ $\leftarrow$ known

if top s entries are different than w^*:

$\text{Max}(e_i) - \text{Min}(e_i) = 2\sigma\sqrt{\log(d)} - (-2\sigma\sqrt{\log(d)})$

So $|e_i| \leq 4\sigma\sqrt{\log(d)}$

3.H. $P(\|\hat{w}_{top(s)} - w^*\|_2^2 \leq 32\sigma^2 s\log(d)) \geq 1 - 1/d$

$= P(\|e\|_2^2 \leq 32\sigma^2 \cdot s \cdot \log(d)) = P(\sum |e_i|^2 \leq 32\sigma^2 s\log(d))$

• e is at most $2s$-sparse so

$\leq P(2s \cdot \max_{i \in [1,\ldots,d]} |e_i|^2 \leq 32\sigma^2 s \cdot \log(d))$

$= P(\max |e_i|^2 \leq 16\sigma^2 \log(d))$

$= P(\max |e_i| \leq 4\sigma(\log(d))^{1/2})$ this implies this

$\Rightarrow P(\max |z_i| \leq 2\sigma\sqrt{\log(d)}) = 1 - \underbrace{P(\max |z_i| > 2\sigma(\log(d))^{1/2})}_{\leq 1/d}$

$\geq 1 - 1/d$ as required.

3.I. $P\left(\frac{1}{n}\|X(\hat{w}_{top(s)} - w^*)\|_2^2 \leq 32\sigma^2 \frac{s\log(d)}{n}\right) \geq 1 - 1/d$

• $\|X(\hat{w}_{top(s)} - w^*)\|_2^2 = \|Xe\|_2^2 = (Xe)^T(Xe) = e^T X^T X e = e^T e = \|e\|_2^2$

$= P\left(\frac{1}{n}\|e\|^2 \leq 32\sigma^2 s\frac{\log(d)}{n}\right) = P(\|e\|^2 \leq 32\sigma^2 s\log(d)) \geq 1 - 1/d$

(From part 3.H)

3.7. given $E\left[\frac{1}{\hat{\lambda}}\|X(\hat{w}_{LS} - w^*)\|_2^2\right] = \sigma^2 \frac{d}{\hat{\lambda}}$

& $E[\|\hat{w}_{LS} - w^*\|_2^2] = \sigma^2 d$

- from prev parts ① $P(\|e\|_2^2 \leq 32\sigma^2 s\log(d)) \geq 1 - 1/d$

w.p $\geq 1 - \frac{1}{d}$ ① ② $P\left(\frac{1}{\hat{\lambda}}\|Xe\|_2^2 \leq 32\sigma^2 \frac{s\log(d)}{\hat{\lambda}}\right) \geq 1 - 1/d$

$E[\|\hat{w}_{top(s)} - w^*\|_2^2] \leq E[32\sigma^2 \log(d)] = 32\sigma^2 s\log(d)$

Similarly w.p $\geq 1 - 1/d$ ②

$E\left[\frac{1}{\hat{\lambda}}\|Xe\|_2^2\right] \leq E\left[32\sigma^2 \frac{s\log(d)}{\hat{\lambda}}\right] = 32\sigma^2 \frac{s\log(d)}{\hat{\lambda}}$

$s \leq \frac{d}{32\log(d)}$ then $\begin{cases} E[\|\hat{w}_{top(s)} - w^*\|_2^2] \leq \sigma^2 d \\ E\left[\frac{1}{\hat{\lambda}}\|\hat{w}_{top(s)} - w^*\|_2^2\right] \leq \sigma^2 \frac{d}{\hat{\lambda}} \end{cases}$

3.k. given that we know the important s Features then $e = \hat{w}_{top(s)} - w^*$ is s-sparse

$\Rightarrow P(\|e\|^2 \leq 16\sigma^2 s\log(d)) = P(\sum |e_i|^2 \leq 16\sigma^2 s\log(d))$

$\leq P(s \max_i |e_i|^2 \leq 16\sigma^2 s\log(d))$

$= P(\max_i |e_i| \leq 4\sigma(\log(d))^{1/2})$

$\geq (1 - 1/d)$ (similar to 3.H.

3.k. Cont.

Now,

- Similar to ii: if $\left[s \leq d / 16 \log(d) \right]$

$$\rightarrow E\left[\frac{1}{n} \| X(\hat{w}_{top(s)} - w^*) \|_2^2 \right] \leq 16\sigma^2 \frac{s \log d}{n} \leq \frac{\sigma^2 d}{n}$$

$$\rightarrow E\left[\| \hat{w}_{top(s)} - w^* \|_2^2 \right] \leq 16\sigma^2 s \log(d) \leq \sigma^2 d$$

- this is lower than the behavior for the sparse $\hat{w}_{top(s)}$, in which we did not know the important s features, derived earlier.

-> SO, if we don't know the important features then we @ least double the variance.

3. Bias and Variance of Sparse Linear Regression

In [6]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

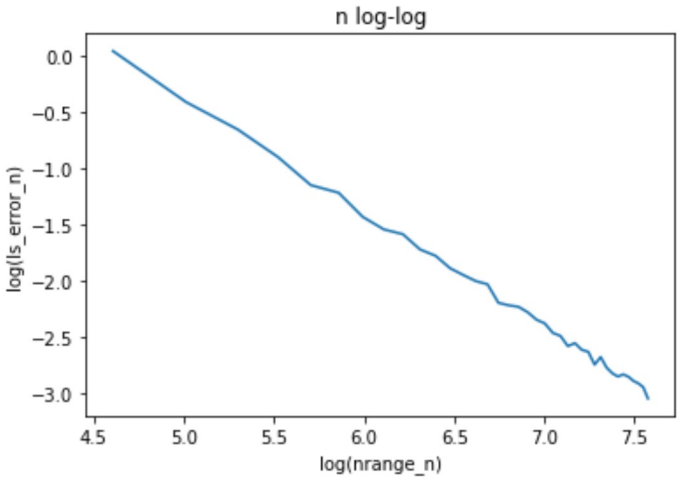
In [7]:
```
def ground_truth(n, d, s):
    """
    Input: Two positive integers n, d. Requires n >= d >=s. If d<s, we let s = d
    Output: A tuple containing i) random matrix of dimension n X d with orthonormal
columns. and
             ii) a d-dimensional, s-sparse wstar with (large) Gaussian entries
    """
    if d > n:
        print("Too many dimensions")
        return None

    if d < s:
        s = d
    A = np.random.randn(n, d) #random Gaussian matrix
    U, S, V = np.linalg.svd(A, full_matrices=False) #reduced SVD of Gaussian matrix
    wstar = np.zeros(d)
    wstar[:s] = 10 * np.random.randn(s)

    np.random.shuffle(wstar)
    return U, wstar

def get_obs(U, wstar):
    """
    Input: U is an n X d matrix and wstar is a d X 1 vector.
    Output: Returns the n-dimensional noisy observation y = U * wstar + z.
    """
    n, d = np.shape(U)
    z = np.random.randn(n) #i.i.d. noise of variance 1
    y = np.dot(U, wstar) + z
    return y
```

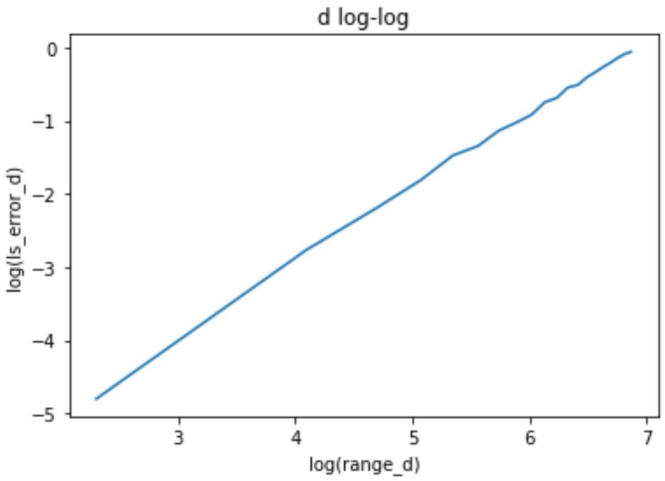
In [8]:
```
def LS(U, y):
    """
    Input: U is an n X d matrix with orthonormal columns and y is an n X 1 vector.
    Output: The OLS estimate what_{LS}, a d X 1 vector.
    """
    wls = np.dot(U.T, y) #pseudoinverse of orthonormal matrix is its transpose
    return wls


def thresh(U, y, lmbda):
    """
    Input: U is an n X d matrix and y is an n X 1 vector; lambda is a scalar thresh
old of the entries.
    Output: The estimate what_{T}(lambda), a d X 1 vector that is hard-thresholded
(in absolute value) at level lambda.
            When code is unfilled, returns the all-zero d-vector.
    """
    n, d = np.shape(U)
    wls = LS(U, y)
    what = np.zeros(d)

    #print np.shape(wls)
    ##########
    #TODO: Fill in thresholding function; store result in what
    ####################
    #YOUR CODE HERE:

    for i in range(len(what)):
        if np.abs(wls[i]) > lmbda:
            what[i] = wls[i]


    ###############
    return what


def topk(U, y, s):
    """
    Input: U is an n X d matrix and y is an n X 1 vector; s is a positive integer.
    Output: The estimate what_{top}(s), a d X 1 vector that has at most s non-zero
entries.
            When code is unfilled, returns the all-zero d-vector.
    """
    n, d = np.shape(U)
    what = np.zeros(d)
    wls = LS(U, y)

    ##########
    #TODO: Fill in thresholding function; store result in what
    ####################
    #YOUR CODE HERE: Remember the absolute value!
    inds = np.argpartition(np.abs(wls), -s)[-s:]
    what[inds] = wls[inds]
    ###############
    return what
```

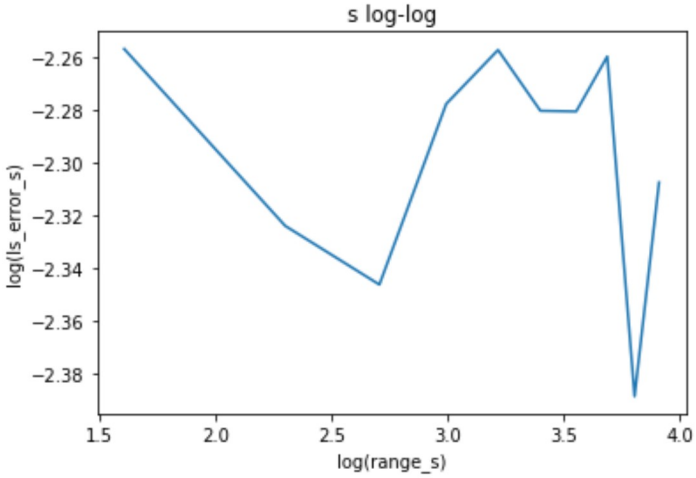
In [9]:
```
def error_calc(num_iters=10, param='n', n=1000, d=100, s=5, s_model=True, true_s=5)
:
    """
    Plots the prediction error 1/n || U(what - wstar)||^2 = 1/n || what - wstar ||^
2 for the three estimators
    averaged over num_iter experiments.

    Input:
    Output: 4 arrays -- range of parameters, errors of LS, topk, and thresh estimat
or, respectively. If thresh and topk
            functions have not been implemented yet, then these errors are simply t
he norm of wstar.
    """
    wls_error = []
    wtopk_error = []
    wthresh_error = []

    if param == 'n':
        arg_range = np.arange(100, 2000, 50)
        lmbda = 2 * np.sqrt(np.log(d))
        for n in arg_range:
            U, wstar = ground_truth(n, d, s) if s_model else ground_truth(n, d, tru
e_s)
            error_wls = 0
            error_wtopk = 0
            error_wthresh = 0
            for count in range(num_iters):
                y = get_obs(U, wstar)
                wls = LS(U, y)
                wtopk = topk(U, y, s)
                wthresh = thresh(U, y, lmbda)
                error_wls += np.linalg.norm(wstar - wls)**2
                error_wtopk += np.linalg.norm(wstar - wtopk)**2
                error_wthresh += np.linalg.norm(wstar - wthresh)**2
            wls_error.append(float(error_wls)/ n / num_iters)
            wtopk_error.append(float(error_wtopk)/ n / num_iters)
            wthresh_error.append(float(error_wthresh)/ n / num_iters)

    elif param == 'd':
        arg_range = np.arange(10, 1000, 50)
        for d in arg_range:
            lmbda = 2 * np.sqrt(np.log(d))
            U, wstar = ground_truth(n, d, s) if s_model else ground_truth(n, d, tru
e_s)
            error_wls = 0
            error_wtopk = 0
            error_wthresh = 0
            for count in range(num_iters):
                y = get_obs(U, wstar)
                wls = LS(U, y)
                wtopk = topk(U, y, s)
                wthresh = thresh(U, y, lmbda)
                error_wls += np.linalg.norm(wstar - wls)**2
                error_wtopk += np.linalg.norm(wstar - wtopk)**2
                error_wthresh += np.linalg.norm(wstar - wthresh)**2
            wls_error.append(float(error_wls)/ n / num_iters)
            wtopk_error.append(float(error_wtopk)/ n / num_iters)
            wthresh_error.append(float(error_wthresh)/ n / num_iters)

    elif param == 's':
        arg_range = np.arange(5, 55, 5)
        lmbda = 2 * np.sqrt(np.log(d))
        for s in arg_range:
            U, wstar = ground_truth(n, d, s) if s_model else ground_truth(n, d, tru
```

a

In [10]:
```
#nrange contains the range of n used, ls_error the corresponding errors for the OLS
estimate
nrange, ls_error, _, _ = error_calc(num_iters=10, param='n', n=1000, d=100, s=5, s_
model=True, true_s=5)
########
#TODO: Your code here: call the helper function for d and s, and plot everything
nrange_d, ls_error_d, _, _ = error_calc(num_iters=10, param='d', n=1000, d=100, s=5
, s_model=True, true_s=5)
nrange_s, ls_error_s, _, _ = error_calc(num_iters=10, param='s', n=1000, d=100, s=5
, s_model=True, true_s=5)

########
#YOUR CODE HERE:

plt.title("n log-log")
plt.plot(np.log(nrange), np.log(ls_error))
plt.xlabel('log(nrange_n)')
plt.ylabel('log(ls_error_n)')
plt.show()

plt.title("d log-log")
plt.plot(np.log(nrange_d), np.log(ls_error_d))
plt.xlabel('log(range_d)')
plt.ylabel('log(ls_error_d)')
plt.show()

plt.title("s log-log")
plt.plot(np.log(nrange_s), np.log(ls_error_s))
plt.xlabel('log(range_s)')
plt.ylabel('log(ls_error_s)')
plt.show()
```


n log-log
log(ls_error_n)
log(nrange_n)
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5


d log-log
log(ls_error_d)
log(range_d)
0
−1
−2
−3
−4
−5
3
4
5
6
7


s log-log
log(ls_error_s)
log(range_s)
−2.26
−2.28
−2.30
−2.32
−2.34
−2.36
−2.38
1.5
2.0
2.5
3.0
3.5
4.0

Are these plots as expected? Discuss. Also put down your parameter choices (either here or in plot captions.) It's fine to use the default values, but put them down nonetheless.

Answer Here

Paramaters for plot of error as function of n:

n=1000, d=100, s=5, s_model=True, true_s=5

Paramaters for plot of error as function of d:

n=1000, d=100, s=5, s_model=True, true_s=5

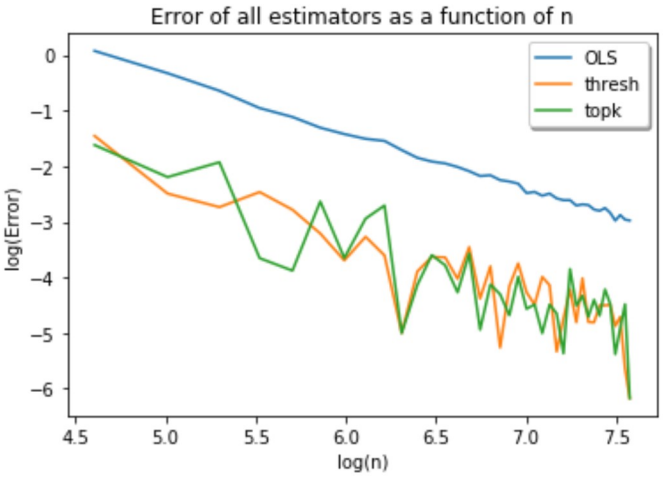
Paramaters for plot of error as function of s:

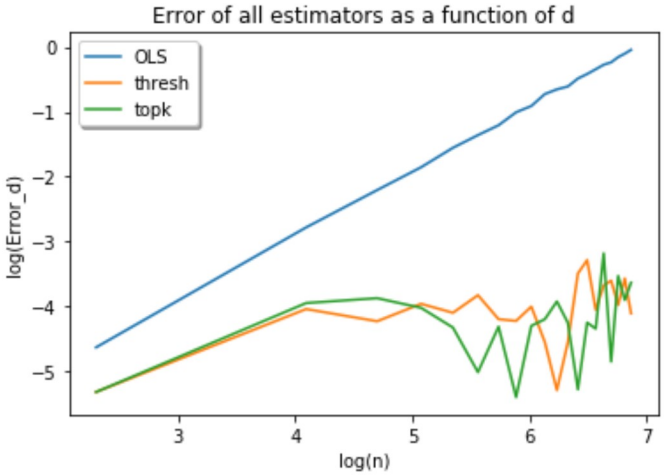
n=1000, d=100, s=5, s_model=True, true_s=5

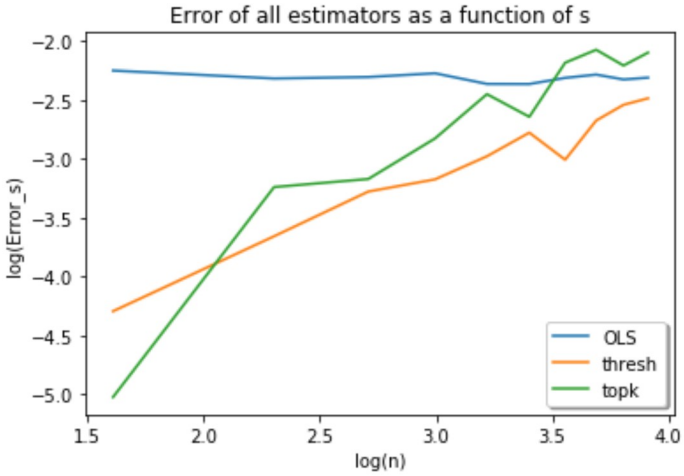
- **First Plot:** Error decreases as we increase the number of data points n. First plot makes intutive sense. The bias decreases as a function of n increasing.
- **Second Plot:** Error increases as the degree increases. The number of data points n is fixed. The complexity increases as the degree of the polynomial increases. Increasing complexity increases the variance which contributes to the increased error.
- **Third Plot:** Error is not correlated with the sparcity.

b

In [17]:
```
#TODO: Part (b)
##############
#YOUR CODE HERE:

nrange, ls_error, thresh_error_n, topk_error_n = error_calc(num_iters=10, param='n'
, n=1000, d=100, s=5, s_model=True, true_s=5)
nrange_d, ls_error_d, thresh_error_d, topk_error_d = error_calc(num_iters=10, param
='d', n=1000, d=100, s=5, s_model=True, true_s=5)
nrange_s, ls_error_s, thresh_error_s, topk_error_s = error_calc(num_iters=10, param
='s', n=1000, d=100, s=5, s_model=True, true_s=5)



plt.title("Error of all estimators as a function of n")
plt.plot(np.log(nrange), np.log(ls_error), label = 'OLS')
plt.plot(np.log(nrange), np.log(thresh_error_n), label = 'thresh')
plt.plot(np.log(nrange), np.log(topk_error_n), label = 'topk')
plt.xlabel('log(n)')
plt.ylabel('log(Error)')
plt.legend(loc='upper right', shadow=True)

plt.show()

plt.title("Error of all estimators as a function of d")
plt.plot(np.log(nrange_d), np.log(ls_error_d), label = 'OLS')
plt.plot(np.log(nrange_d), np.log(thresh_error_d), label = 'thresh')
plt.plot(np.log(nrange_d), np.log(topk_error_d), label = 'topk')
plt.xlabel('log(n)')
plt.ylabel('log(Error_d)')
plt.legend(loc='upper left', shadow=True)



plt.show()


plt.title("Error of all estimators as a function of s")
plt.plot(np.log(nrange_s), np.log(ls_error_s), label = 'OLS')
plt.plot(np.log(nrange_s), np.log(thresh_error_s), label = 'thresh')
plt.plot(np.log(nrange_s), np.log(topk_error_s), label = 'topk')
plt.xlabel('log(n)')
plt.ylabel('log(Error_s)')
plt.legend(loc='lower right', shadow=True)



plt.show()
```

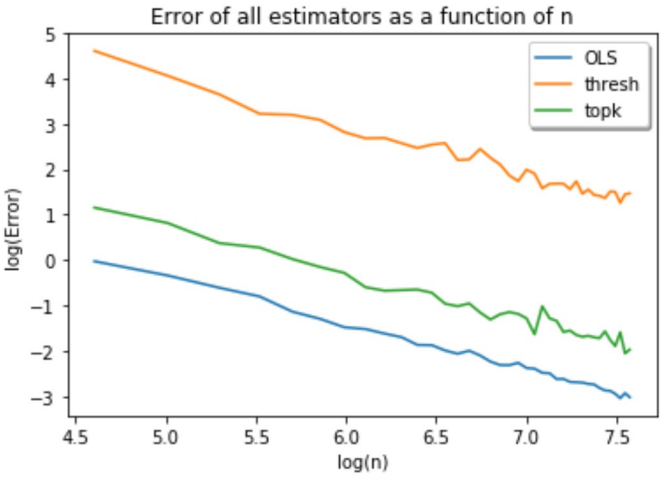

Error of all estimators as a function of n
OLS
thresh
topk
log(Error)
log(n)

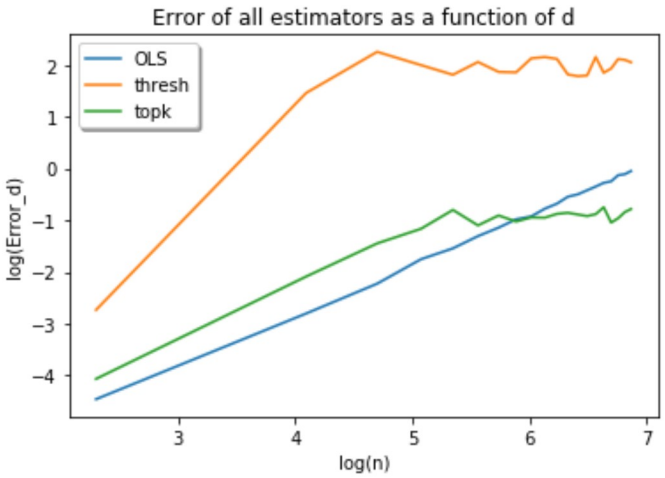

Error of all estimators as a function of d
OLS
thresh
topk
log(Error_d)
log(n)

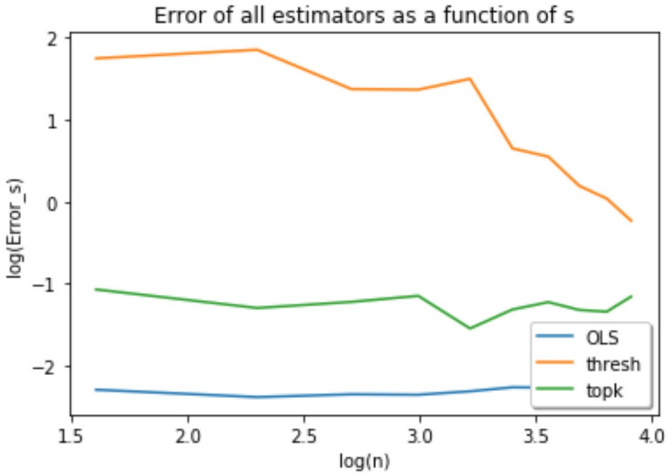

Error of all estimators as a function of s
OLS
thresh
topk
log(Error_s)
log(n)

c

In [18]:
```
#TODO: Part (c)
##############
#YOUR CODE HERE:
nrange, ls_error, thresh_error_n, topk_error_n = error_calc(num_iters=10, param='n'
, n=1000, d=100, s=5, s_model=False, true_s=100)
nrange_d, ls_error_d, thresh_error_d, topk_error_d = error_calc(num_iters=10, param
='d', n=1000, d=100, s=5, s_model=False, true_s=100)
nrange_s, ls_error_s, thresh_error_s, topk_error_s = error_calc(num_iters=10, param
='s', n=1000, d=100, s=5, s_model=False, true_s=100)


plt.title("Error of all estimators as a function of n")
plt.plot(np.log(nrange), np.log(ls_error), label = 'OLS')
plt.plot(np.log(nrange), np.log(thresh_error_n), label = 'thresh')
plt.plot(np.log(nrange), np.log(topk_error_n), label = 'topk')
plt.xlabel('log(n)')
plt.ylabel('log(Error)')
plt.legend(loc='upper right', shadow=True)

plt.show()

plt.title("Error of all estimators as a function of d")
plt.plot(np.log(nrange_d), np.log(ls_error_d), label = 'OLS')
plt.plot(np.log(nrange_d), np.log(thresh_error_d), label = 'thresh')
plt.plot(np.log(nrange_d), np.log(topk_error_d), label = 'topk')
plt.xlabel('log(n)')
plt.ylabel('log(Error_d)')
plt.legend(loc='upper left', shadow=True)



plt.show()


plt.title("Error of all estimators as a function of s")
plt.plot(np.log(nrange_s), np.log(ls_error_s), label = 'OLS')
plt.plot(np.log(nrange_s), np.log(thresh_error_s), label = 'thresh')
plt.plot(np.log(nrange_s), np.log(topk_error_s), label = 'topk')
plt.xlabel('log(n)')
plt.ylabel('log(Error_s)')
plt.legend(loc='lower right', shadow=True)



plt.show()
```


Error of all estimators as a function of n
OLS
thresh
topk
log(Error)
log(n)


Error of all estimators as a function of d
OLS
thresh
topk
log(Error_d)
log(n)


Error of all estimators as a function of s
OLS
thresh
topk
log(Error_s)
log(n)

Param First Plot

error_calc(num_iters=10, param='n', n=1000, d=100, s=5, s_model=False, true_s=100)

Param Second Plot

error_calc(num_iters=10, param='d', n=1000, d=100, s=5, s_model=False, true_s=100)

Param Third Plot

error_calc(num_iters=10, param='s', n=1000, d=100, s=5, s_model=False, true_s=100)

Answer Here

- **First Plot:** Error as a function of n decreases as n increases. The bias decreases. For data from a non-sparse linear model: Same statements made earlier regarding the general behavior of the OLS plots are true here for for the plotted error of the estimators and the OLS as a function of n.
- **Second Plot:** For data from a non-sparse linear model: Same statements made earlier regarding the general behavior of the OLS plots are true here for the plotted error of the estimators and the OLS as a function of d.

From Part a

"Error increases as the degree increases. The number of data points n is fixed. The complexity increases as the degree of the polynomial increases. Increasing complexity increases the variance which contributes to the increased error. "

- **Third Plot:** As we increase the sparsity, we note that this reduces the error as the sparsity of the true model approaches the sparcity our estimators assume the data to have.

The true sparsity, s, of the model is greater than the s-sparsity assumed for our estimators. Our estimators want sparse data, the estimators will incorrectly estimate at most (true_s - s) weights in w^{hat} as zero when they are in fact non-zero/non-sparse coefficients of the weight vector. Thus our estimates have worse performance as they are assuming sparsity in the data when the true data is generated from a non-sparse linear model.

4. Decision Trees and Random Forests

In [7]:
```
%matplotlib inline
```

a)

In [28]:
```
# You may want to install "gprof2dot"
import io
from collections import Counter

import numpy as np
import scipy.io
import sklearn.model_selection
import sklearn.tree
from numpy import genfromtxt
from scipy import stats
from sklearn.base import BaseEstimator, ClassifierMixin

import pydot

eps = 1e-5  # a small number

class DecisionTree:
    def __init__(self, max_depth=3, feature_labels=None):
        self.max_depth = max_depth
        self.features = feature_labels
        self.left, self.right = None, None  # for non-leaf nodes
        self.split_idx, self.thresh = None, None  # for non-leaf nodes
        self.data, self.pred = None, None  # for leaf nodes

    @staticmethod
    def information_gain(X, y, thresh):
        # TODO implement information gain function
        def entropy(ys):
            total = len(ys)
            if total == 0:
                return 0
            arr = [0, 0]
            for label in ys:
                arr[label] += 1
            return sum([-(pi/total)*np.log((pi/total)) for pi in arr if pi != 0])
        total = len(y)
        y_left = []
        y_right = []
        for i in range(total):
            if X[i] < thresh:
                y_left.append(y[i])
            else:
                y_right.append(y[i])
        return entropy(y) - (len(y_left)/total)*entropy(y_left) - (len(y_right)/tot
al)*entropy(y_right)

    @staticmethod
    def gini_purification(X, y, thresh):
        # TODO implement gini_purification function
        def gini(ys):
            total = len(ys)
            if total == 0:
                return 0
            arr = [0, 0]
            for label in ys:
                arr[label] += 1
            return 1 - sum([(pi/total)**2 for pi in arr])
        total = len(y)
        y_left = []
        y_right = []
        for i in range(total):
            if X[i] < thresh:
                y_left.append(y[i])
            else:
```

b)

All answers can be found in preprocess function.

- Some data points are misssing class labels;

  # Temporarily assign -1 to missing data

- Some features are not numerical values;

  # Hash the columns (used for handling strings)

- Some data points are missing some features.

  # Replace missing data with the mode value. We use the mode instead of
  # the mean or median because this makes more sense for categorical
  # features such as gender or cabin type, which are not ordered.

Titanic Dataset

In [29]:
```
if __name__ == "__main__":
    main("titanic")
```

```
/home/nlorio/.local/lib/python3.5/site-packages/ipykernel_launcher.py:287: Visib
leDeprecationWarning: Reading unicode strings without specifying the encoding ar
gument is deprecated. Set the encoding, use None for the system default.
/home/nlorio/.local/lib/python3.5/site-packages/ipykernel_launcher.py:289: Visib
leDeprecationWarning: Reading unicode strings without specifying the encoding ar
gument is deprecated. Set the encoding, use None for the system default.


Part (b): preprocessing the titanic dataset
Features: [b'pclass', b'sex', b'age', b'sibsp', b'parch', b'ticket', b'fare', b'
cabin', b'embarked', b'male', b'female', b'S', b'C', b'Q']
Train/test size: (999, 14) (310, 14)


Part 0: constant classifier
Accuracy 0.6136136136136137


Part (a-b): simplified decision tree
Predictions [1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 0. 1
. 1.
 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0.
 1. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 1. 0. 1. 0.
 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0.
 1. 0. 0. 0.]


Part (c): sklearn's decision tree
Cross validation [0.79640719 0.76576577 0.79819277]


Part (e): Our Bagged Tree for titanic
Cross validation [0.79041916 0.78378378 0.8253012 ]
First splits [(b'male', 101), (b'female', 99)]
BaggedTrees(n=200, params={'max_depth': 5, 'min_samples_leaf': 10})


Part (g): Our RandomForest for titanic
Cross validation [0.7994012  0.77477477 0.81927711]
First splits [(b'age', 24), (b'pclass', 23), (b'male', 23), (b'S', 22), (b'parch
', 20), (b'fare', 20), (b'C', 20), (b'female', 18), (b'sibsp', 15), (b'Q', 15)]
RandomForest(m=None, n=200, params={'max_depth': 5, 'min_samples_leaf': 10})


Part (i, j): Our RandomForest for titanic
Cross validation [0.81137725 0.75375375 0.74698795]
First splits [(b'age', 24), (b'pclass', 23), (b'male', 23), (b'S', 22), (b'parch
', 20), (b'fare', 20), (b'C', 20), (b'female', 18), (b'sibsp', 15), (b'Q', 15)]
BoostedRandomForest(m=None, n=200,
          params={'max_depth': 5, 'min_samples_leaf': 10})
[1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1
, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0,
0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1,
1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0,
0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0,
0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0,
1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0]
```

Spam Dataset

In [30]:
```
if __name__ == "__main__":
    main(dataset="spam")
```

```
Features: ['pain', 'private', 'bank', 'money', 'drug', 'spam', 'prescription', '
creative', 'height', 'featured', 'differ', 'width', 'other', 'energy', 'business
', 'message', 'volumes', 'revision', 'path', 'meter', 'memo', 'planning', 'pleas
ed', 'record', 'out', 'semicolon', 'dollar', 'sharp', 'exclamation', 'parenthesi
s', 'square_bracket', 'ampersand']
Train/test size: (5172, 32) (5857, 32)


Part 0: constant classifier
Accuracy 0.7099767981438515


Part (a-b): simplified decision tree
Predictions [0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0
. 0.
 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 1.
 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0.
 1. 1. 0. 1.]


Part (c): sklearn's decision tree
Cross validation [0.80104408 0.80800464 0.77900232]


Part (e): Our Bagged Tree for spam
Cross validation [0.80220418 0.80916473 0.78538283]
First splits [('exclamation', 200)]
BaggedTrees(n=200, params={'max_depth': 5, 'min_samples_leaf': 10})


Part (g): Our RandomForest for spam
Cross validation [0.79988399 0.80858469 0.78480278]
First splits [('money', 10), ('drug', 10), ('path', 10), ('semicolon', 10), ('pa
renthesis', 10), ('other', 9), ('memo', 9), ('featured', 8), ('pain', 7), ('priv
ate', 7), ('bank', 7), ('prescription', 7), ('differ', 7), ('business', 7), ('vo
lumes', 7), ('meter', 7), ('ampersand', 7), ('creative', 6), ('record', 6), ('ou
t', 6), ('dollar', 6), ('exclamation', 6), ('width', 5), ('square_bracket', 5),
('energy', 4), ('revision', 4), ('pleased', 4), ('sharp', 4), ('spam', 2), ('hei
ght', 1), ('message', 1), ('planning', 1)]
RandomForest(m=None, n=200, params={'max_depth': 5, 'min_samples_leaf': 10})


Part (i, j): Our RandomForest for spam
Cross validation [0.82540603 0.84570766 0.7900232 ]
First splits [('money', 10), ('semicolon', 10), ('parenthesis', 10), ('square_br
acket', 10), ('other', 9), ('path', 9), ('drug', 8), ('memo', 8), ('pain', 7), (
'private', 7), ('bank', 7), ('prescription', 7), ('differ', 7), ('business', 7),
('volumes', 7), ('meter', 7), ('ampersand', 7), ('creative', 6), ('record', 6),
('out', 6), ('dollar', 6), ('exclamation', 6), ('featured', 5), ('square_bracket
', 5), ('energy', 4), ('revision', 4), ('pleased', 4), ('sharp', 4), ('width', 3
), ('spam', 2), ('message', 1), ('planning', 1)]
BoostedRandomForest(m=None, n=200,
          params={'max_depth': 5, 'min_samples_leaf': 10})
[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1
, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1,
0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
```

h)

Part H Observations:

Qualitatively describe what this algorithm is doing. What does it mean when a_i < 0, how does the algorithm handle such trees?

Boosting:

Fit:

- We determine the weighted error of the current iteration decision tree. a_i is computed based off of this weighted error in accordance with AdaBoost algorithm.
- a_i is always >= 0. The sign assigned to a_i depends on whether or not its respective data point was correctly classified by the current iteration

For data points that are correctly classified we lower the precedence of said data point by reducing its weight, wi ,by a factor of exp(-a_i). Vice a versa for incorreclty classified data points, it increases the weight wi on the data point by exp(+a_i). The new weights on the data points are used in the next iteration.

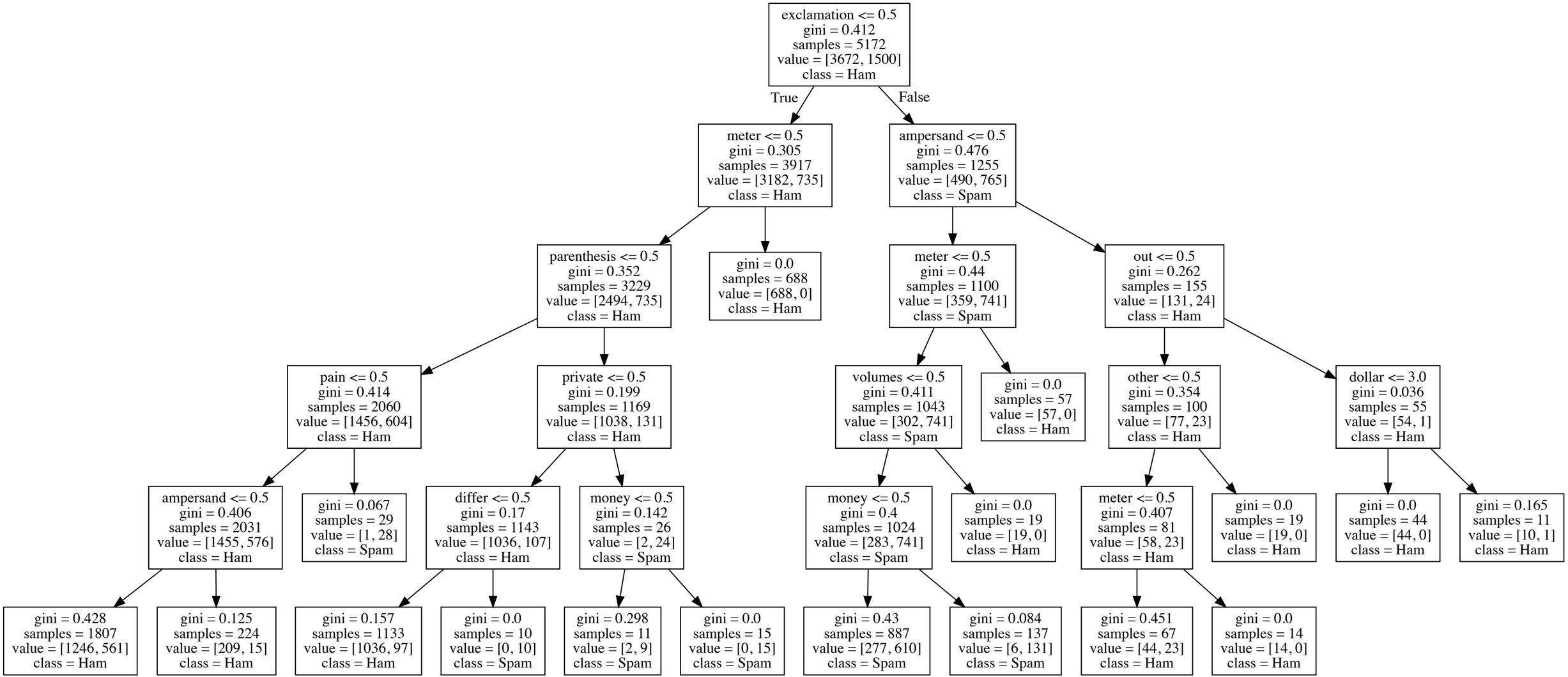
Prediction:

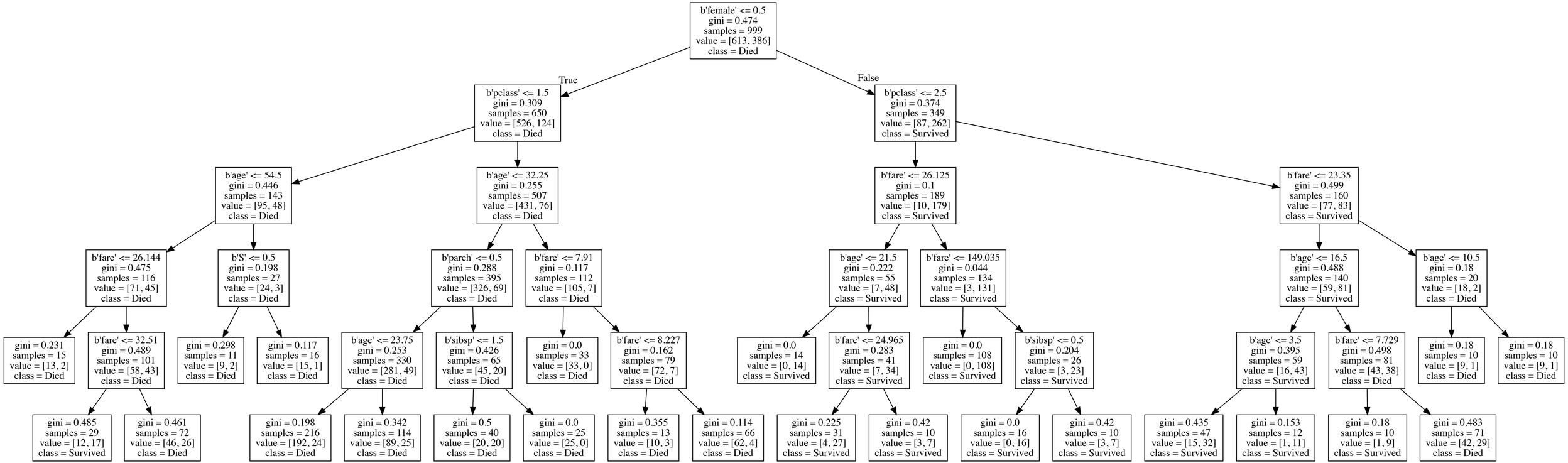
The performance of the classification in the current iteration is determined by the score.

Our algorithm is repeated until we have M trees and have fit and predicted our classification M times, with updated weights at each iteration.

i)

- Most challenging data to classify with decision trees are : Data with high variance, as for decision trees small variations in the data might result in a completely different tree being generated. Furthermore if some classes of data dominate than decision tree learners can create biased trees.


exclamation <= 0.5
gini = 0.412
samples = 5172
value = [3672, 1500]
class = Ham
True
False
meter <= 0.5
gini = 0.305
samples = 3917
value = [3182, 735]
class = Ham
ampersand <= 0.5
gini = 0.476
samples = 1255
value = [490, 765]
class = Spam
parenthesis <= 0.5
gini = 0.352
samples = 3229
value = [2494, 735]
class = Ham
gini = 0.0
samples = 688
value = [688, 0]
class = Ham
meter <= 0.5
gini = 0.44
samples = 1100
value = [359, 741]
class = Spam
out <= 0.5
gini = 0.262
samples = 155
value = [131, 24]
class = Ham
pain <= 0.5
gini = 0.414
samples = 2060
value = [1456, 604]
class = Ham
private <= 0.5
gini = 0.199
samples = 1169
value = [1038, 131]
class = Ham
volumes <= 0.5
gini = 0.411
samples = 1043
value = [302, 741]
class = Spam
gini = 0.0
samples = 57
value = [57, 0]
class = Ham
other <= 0.5
gini = 0.354
samples = 100
value = [77, 23]
class = Ham
dollar <= 3.0
gini = 0.036
samples = 55
value = [54, 1]
class = Ham
ampersand <= 0.5
gini = 0.406
samples = 2031
value = [1455, 576]
class = Ham
gini = 0.067
samples = 29
value = [1, 28]
class = Spam
differ <= 0.5
gini = 0.17
samples = 1143
value = [1036, 107]
class = Ham
money <= 0.5
gini = 0.142
samples = 26
value = [2, 24]
class = Spam
money <= 0.5
gini = 0.4
samples = 1024
value = [283, 741]
class = Spam
gini = 0.0
samples = 19
value = [19, 0]
class = Ham
meter <= 0.5
gini = 0.407
samples = 81
value = [58, 23]
class = Ham
gini = 0.0
samples = 19
value = [19, 0]
class = Ham
gini = 0.0
samples = 44
value = [44, 0]
class = Ham
gini = 0.165
samples = 11
value = [10, 1]
class = Ham
gini = 0.428
samples = 1807
value = [1246, 561]
class = Ham
gini = 0.125
samples = 224
value = [209, 15]
class = Ham
gini = 0.157
samples = 1133
value = [1036, 97]
class = Ham
gini = 0.0
samples = 10
value = [0, 10]
class = Spam
gini = 0.298
samples = 11
value = [2, 9]
class = Spam
gini = 0.0
samples = 15
value = [0, 15]
class = Spam
gini = 0.43
samples = 887
value = [277, 610]
class = Spam
gini = 0.084
samples = 137
value = [6, 131]
class = Spam
gini = 0.451
samples = 67
value = [44, 23]
class = Ham
gini = 0.0
samples = 14
value = [14, 0]
class = Ham


b'female' <= 0.5
gini = 0.474
samples = 999
value = [613, 386]
class = Died
True
False
b'pclass' <= 1.5
gini = 0.309
samples = 650
value = [526, 124]
class = Died
b'pclass' <= 2.5
gini = 0.374
samples = 349
value = [87, 262]
class = Survived
b'age' <= 54.5
gini = 0.446
samples = 143
value = [95, 48]
class = Died
b'age' <= 32.25
gini = 0.255
samples = 507
value = [431, 76]
class = Died
b'fare' <= 26.125
gini = 0.1
samples = 189
value = [10, 179]
class = Survived
b'fare' <= 23.35
gini = 0.499
samples = 160
value = [77, 83]
class = Survived
b'fare' <= 26.144
gini = 0.475
samples = 116
value = [71, 45]
class = Died
b'S' <= 0.5
gini = 0.198
samples = 27
value = [24, 3]
class = Died
b'parch' <= 0.5
gini = 0.288
samples = 395
value = [326, 69]
class = Died
b'fare' <= 7.91
gini = 0.117
samples = 112
value = [105, 7]
class = Died
b'age' <= 21.5
gini = 0.222
samples = 55
value = [7, 48]
class = Survived
b'fare' <= 149.035
gini = 0.044
samples = 134
value = [3, 131]
class = Survived
b'age' <= 16.5
gini = 0.488
samples = 140
value = [59, 81]
class = Survived
b'age' <= 10.5
gini = 0.18
samples = 20
value = [18, 2]
class = Died
gini = 0.231
samples = 15
value = [13, 2]
class = Died
b'fare' <= 32.51
gini = 0.489
samples = 101
value = [58, 43]
class = Died
gini = 0.298
samples = 11
value = [9, 2]
class = Died
gini = 0.117
samples = 16
value = [15, 1]
class = Died
b'age' <= 23.75
gini = 0.253
samples = 330
value = [281, 49]
class = Died
b'sibsp' <= 1.5
gini = 0.426
samples = 65
value = [45, 20]
class = Died
gini = 0.0
samples = 33
value = [33, 0]
class = Died
b'fare' <= 8.227
gini = 0.162
samples = 79
value = [72, 7]
class = Died
gini = 0.0
samples = 14
value = [0, 14]
class = Survived
b'fare' <= 24.965
gini = 0.283
samples = 41
value = [7, 34]
class = Survived
gini = 0.0
samples = 108
value = [0, 108]
class = Survived
b'sibsp' <= 0.5
gini = 0.204
samples = 26
value = [3, 23]
class = Survived
b'age' <= 3.5
gini = 0.395
samples = 59
value = [16, 43]
class = Survived
b'fare' <= 7.729
gini = 0.498
samples = 81
value = [43, 38]
class = Died
gini = 0.18
samples = 10
value = [9, 1]
class = Died
gini = 0.18
samples = 10
value = [9, 1]
class = Died
gini = 0.485
samples = 29
value = [12, 17]
class = Survived
gini = 0.461
samples = 72
value = [46, 26]
class = Died
gini = 0.198
samples = 216
value = [192, 24]
class = Died
gini = 0.342
samples = 114
value = [89, 25]
class = Died
gini = 0.5
samples = 40
value = [20, 20]
class = Died
gini = 0.0
samples = 25
value = [25, 0]
class = Died
gini = 0.355
samples = 13
value = [10, 3]
class = Died
gini = 0.114
samples = 66
value = [62, 4]
class = Died
gini = 0.225
samples = 31
value = [4, 27]
class = Survived
gini = 0.42
samples = 10
value = [3, 7]
class = Survived
gini = 0.0
samples = 16
value = [0, 16]
class = Survived
gini = 0.42
samples = 10
value = [3, 7]
class = Survived
gini = 0.435
samples = 47
value = [15, 32]
class = Survived
gini = 0.153
samples = 12
value = [1, 11]
class = Survived
gini = 0.18
samples = 10
value = [1, 9]
class = Survived
gini = 0.483
samples = 71
value = [42, 29]
class = Died
gini = 0.18
samples = 10
value = [9, 1]
class = Died
gini = 0.18
samples = 10
value = [9, 1]
class = Died

We see that boosted trees perform best and this is due to the fact they iteratively learn and can generate better future iterations. Boosting is based on weak learners with high bias and low variance. Random forests are based on low bias and high variance. Bias in our case performs better