CS 189: Introduction to Machine Learning

Fall 2017

●

Homework 12

●

Solutions by

Jinhong Du

3033483677

Question 1

(a)

Jinhong Du
jaydu@berkeley.edu

(b)

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. Jinhong Du

Question 2

(a)

$\because$

$$A^T A = I$$

$\therefore$

$$A_i^T A_j = \delta_{ij}$$

$\because$

$$Aw = \begin{pmatrix} \sum_{j=1}^{d} a_{1j} w_j \\ \sum_{j=1}^{d} a_{2j} w_j \\ \vdots \\ \sum_{j=1}^{d} a_{nj} w_j \end{pmatrix}$$

$$= \sum_{j=1}^{d} A_j w_j$$

$\therefore$

$$\begin{aligned} J_\lambda(w) &= \frac{1}{2}\|y - Aw\|_2^2 + \lambda\|w\|_1 \\ &= \frac{1}{2}\left\| y - \sum_{j=1}^{d} A_j w_j \right\|_2^2 + \lambda\|w\|_1 \\ &= \frac{1}{2}\left(y - \sum_{j=1}^{d} A_j w_j \right)^T \left(y - \sum_{j=1}^{d} A_j w_j \right) + \lambda\|w\|_1 \\ &= \frac{1}{2}y^T y - \sum_{j=1}^{d} y^T A_j w_j + \frac{1}{2}\sum_{j=1}^{d} A_j^T w_j \left(\sum_{j=1}^{d} A_j w_j \right) + \lambda\|w\|_1 \\ &= \frac{1}{2}y^T y - \sum_{j=1}^{d} y^T A_j w_j + \frac{1}{2}\sum_{j=1}^{d} w_j^2 + \sum_{i=1}^{d} \lambda|w_i| \\ &= \frac{1}{2}y^T y + \sum_{i=1}^{d} \left(-y^T A_i w_i + \frac{1}{2}w_i^2 + \lambda|w_i| \right) \end{aligned}$$

$\therefore$

$$g(y) = \frac{1}{2}y^T y$$

and

$$f(A_i, y, w_i, \lambda) = -y^T A_i w_j + \frac{1}{2}w_i^2 + \lambda|w_i|$$

$\therefore$

$$\min_w J_\lambda(w) = \sum_{i=1}^{d} \min_{w_i} f(A_i, y, w_i, \lambda)$$

i.e. w_i^* is determined by the i-th feature and the output regardless of other features

(b)

When $w_i^* > 0$, let

$$\begin{aligned}\frac{\partial}{\partial w_i} f(A_i, y, w_i, \lambda) &= \frac{\partial}{\partial w_i}\left(-y^T A_i w_i + \frac{1}{2}w_i^2 + \lambda w_i\right)\\ &= -y^T A_i + w_i + \lambda\\ &= 0\end{aligned}$$

we have

$$w_i^* = y^T A_i - \lambda$$

(c)

When $w_i^* < 0$, let

$$\begin{aligned}\frac{\partial}{\partial w_i} f(A_i, y, w_i, \lambda) &= \frac{\partial}{\partial w_i}\left(-y^T A_i w_i + \frac{1}{2}w_i^2 - \lambda w_i\right)\\ &= -y^T A_i + w_i - \lambda\\ &= 0\end{aligned}$$

we have

$$w_i^* = y^T A_i + \lambda$$

(d)

Since when $w_i^* > 0$,

$$w_i^* = y^T A_i - \lambda$$

when $w_i^* < 0$,

$$w_i^* = y^T A_i + \lambda$$

we have the condition for w_i^* to be zero is

$$\begin{cases} y^T A_i - \lambda \leqslant 0 & , w_i^* > 0\\ y^T A_i + \lambda \geqslant 0 & , w_i^* < 0\end{cases}$$

(e)

Similar to (a), we have

$$\begin{aligned}J_\lambda(w) &= \frac{1}{2}y^T y + \sum_{i=1}^{d}\left(-y^T A_i w_i + \frac{1}{2}w_i^2 + \lambda w_i^2\right)\\ &= g(y) + \sum_{i=1}^{d} f(A_i, y, w_i, \lambda)\end{aligned}$$

Solution (cont.)

Let

$$\begin{aligned}\frac{\partial}{\partial w_i} f(A_i, y, w_i, \lambda) &= \frac{\partial}{\partial w_i}\left(-y^T A_i w_i + \frac{1}{2} w_i^2 + \lambda w_i^2\right) \\ &= -y^T A_i + w_i + 2\lambda w_i \\ &= 0\end{aligned}$$

we have

$$w_i^* = \frac{y^T A_i}{2\lambda + 1}$$

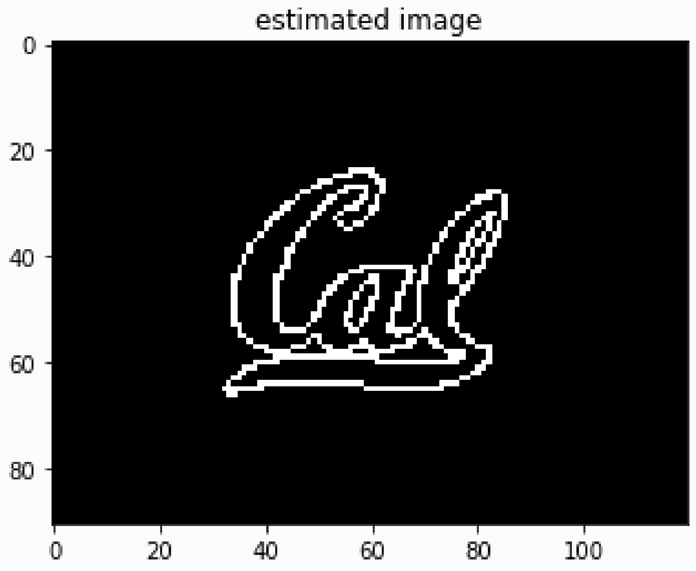
Therefore the condition for $w_i^* = 0$ is

$$y^T A_i = 0$$

It is less likely to satisfy this condition than the one in (d), i.e. w_i^* in l_1 norm is more likely to be zero, i.e. l_1 norm promotes sparsity.

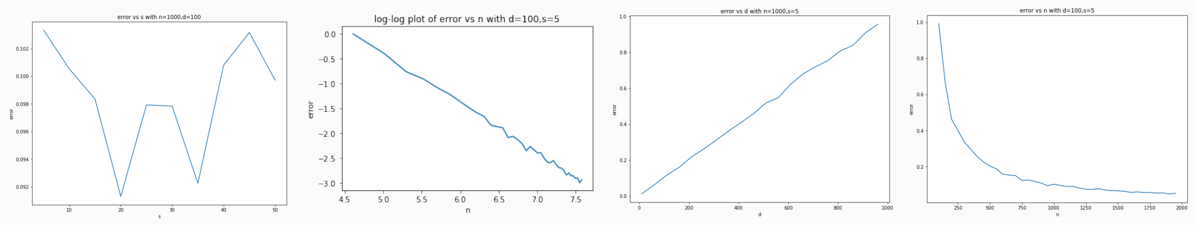
(f)

I have tried $0.000001, 0.000001, 0.0001, 0.0000015, 0.0000001, 0.0000005, 0.00000001$. The best is 0.0000001.



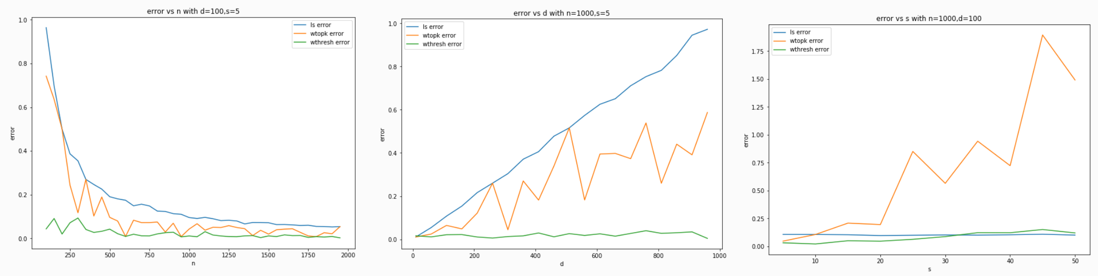
Question 3

(a)

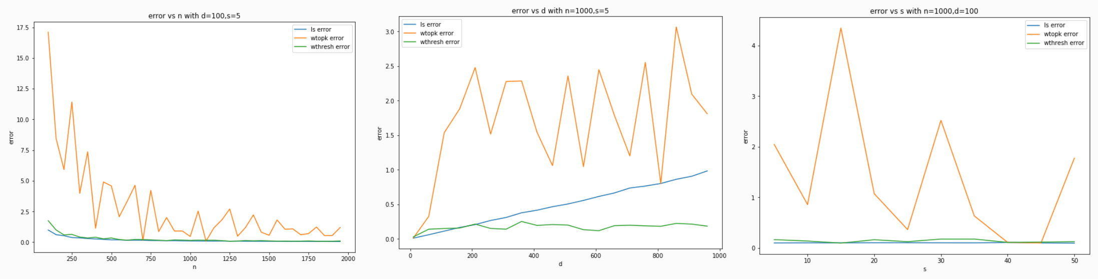


Yes, in the plot, the prediction error seems to have linear relationship with $\frac{1}{n}$ and d respectively. However, the prediction error seems not to have linear relationship with s.

(b)



(c)



Increasing data n helps to reduce the variance just like in plot 1.
Increasing bad features d will increase variance just like in plot 2.
Enforcing sparsity is equivalent to deliberately constraining model complexity. So the variance tends to increase when $\frac{s}{true\ s}$ increase. Just like in plot 3, $true\ s = 50$, so when s increases, the model is more complex and therefore the variance will decrease.

(d)

$$\because \quad Z_1, \cdots, Z_d \sim N(0, \sigma^2)$$

Solution (cont.)

$\therefore$

$$\begin{aligned}\Pr\left\{\max_{i\in\{1,2,\ldots,d\}}|Z_i|\geq 2\sigma\sqrt{\log d}\right\} &\leqslant \Pr\left\{|Z_{i_0}|\geq 2\sigma\sqrt{\log d}, i_0\in\{1,2,\cdots,d\}\right\}\\ &\leqslant e^{-\frac{(2\sigma\sqrt{\log d})}{2\sigma^2}}\\ &= e^{-\log d}\\ &= \frac{1}{d}\end{aligned}$$

(e)

$\because$

$$y = Aw^* + z$$

$$z \sim N(0,\sigma^2 I_n)$$

and

$$z_i \perp z_j (i\neq j)$$

$\therefore$

$$y \sim N(Aw^*,\sigma^2 I_n)$$

and

$$y_i \perp y_j (i\neq j)$$

$\because$

$$\begin{aligned}\hat{w}_{LS} &= (A^TA)^{-1}A^Ty\\ &= A^Ty\\ &= A^T(Aw^*+z)\\ &= w^* + A^Tz\end{aligned}$$

and A has orthonormal columns, i.e. $A^TA = I_d$.

$\therefore$

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

where $z' = A^Tz$ is also Gaussian with variance σ^2 since linear transformation of an normal vector also gives an normal vector, $z' \sim N(A^T0, A^T\sigma^2 I_n A) = N(0,\sigma^2 I_d)$ and $z_i'(i=1,\cdots,d)$ are independent since they are uncorrelated.

$\because$

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

$\therefore$ $\hat{w}_{top}(s)$ returns the top s entries of $w + z'$ measured in their absolute value

(f)

Since $\hat{w}_{top}(s)$ returns the top s entries of $w+z'$ measured in their absolute value and 0 otherwise, it is at most s-sparse, i.e. there are at most s entries of $\hat{w}_{top}(s)$ will be non-zero. And w^* also has at most

Solution (cont.)

s entries be non-zero. Therefore, when the indexes of these non-zero entries are different, e will have at most $2s$ non-zero entries, i.e., e is (at most) $2s$-sparse.

(g)

Since e is at most $2s$-sparse, only at most $2s$ entries of e are non-zeros. So we only consider these entries. If $\hat{w}_{top}(s)_i \neq 0$, then

$$\begin{aligned}|e_i| &\leqslant |\hat{w}_{top}(s)_i - w_i^*| \\ &= |[\hat{w}_{top}(s) - w^*]_i| \\ &= |z_i'| \\ &\leqslant 2\sigma\sqrt{\log d}\end{aligned}$$

If $\hat{w}_{top}(s)_i = 0$, then $w_i^* \neq 0$. Because w^* and $\hat{w}_{top}(w)$ are s-sparse, it means that $|w_i^* + z_i'| \leqslant |0 + z_j'|$ for some j. Otherwise, s non-zero entries of $\hat{w}_{top}(s)$ should be the same indexes of w^*.

$$\begin{aligned}|e_i| &= |w_i^*| \\ &= |w_i^* + z_i' - z_i'| \\ &= |w_i^* + z_i'| + |z_i'| \\ &\leqslant |z_j'| + |z_i'| \\ &\leqslant 4\sigma\sqrt{\log d}\end{aligned}$$

(h)

Since e is at most $2s$-sparse, only at most $2s$ entries of e are non-zeros. So for these non-zero entries, $\because$

$$\Pr\left\{\max_{i\in\{1,2,\ldots,d\}} |z_i'| \geq 2\sigma\sqrt{\log d}\right\} \leqslant \frac{1}{d}$$

$\therefore$

$$\Pr\left\{\max_{i\in\{1,2,\ldots,d\}} |z_i'|^2 \leq 4\sigma^2\log d\right\} \leqslant 1 - \frac{1}{d}$$

$\because$ from (g) we have

$$|e_i| \leqslant 4\sigma\sqrt{\log d}$$

and from (f) we have e is at most $2s$-sparse
$\therefore$

$$\begin{aligned}\|\hat{w}_{top}(s) - w^*\|_2^2 &= \|e\|_2^2 \\ &\leqslant 2s \cdot (2\max_i |z_i'|)^2 \\ &\leqslant 8s\max_i |z_i'|^2\end{aligned}$$

Solution (cont.)

$\therefore$

$$\begin{aligned}\Pr\left\{\|\hat{w}_{top}(s)-w^*\|_2^2 \le 32s\sigma^2\log d\right\} &= \Pr\left\{\frac{\|e\|_2^2}{8s} \le 4\sigma^2\log d\right\}\\ &\geqslant \Pr\left\{\max_{i\in\{1,2,\ldots,d\}} |z_i'|^2 \le 4\sigma^2\log d\right\}\\ &\geqslant 1-\frac{1}{d}\end{aligned}$$

i.e. with probability $1-\frac{1}{d}$,

$$\|\hat{w}_{top}(s)-w^*\|_2^2 \le 32s\sigma^2\log d$$

(i)

With probability at least $1-\frac{1}{d}$,we have

$$\begin{aligned}\frac{1}{n}\|A(\hat{w}_{top}(s)-w^*)\|_2^2 &= \frac{1}{n}(\hat{w}_{top}(s)-w^*)^T A^T A(\hat{w}_{top}(s)-w^*)\\ &= \frac{1}{n}(\hat{w}_{top}(s)-w^*)^T(\hat{w}_{top}(s)-w^*)\\ &= \frac{1}{n}\|\hat{w}_{top}(s)-w^*)\|_2^2\\ &\leqslant 32\sigma^2 s\frac{\log d}{n}\end{aligned}$$

$\because$ sample variance is

$$\begin{aligned}\frac{1}{n}\|\hat{y}_{top}-y\|_2^2 &= \frac{1}{n}\|A\hat{w}_{top}+z-(Aw^*+z)\|_2^2\\ &= \frac{1}{n}\|A(\hat{w}_{top}(s)-w^*)\|_2^2\end{aligned}$$

$\therefore$ $\forall\ \epsilon>0,\ \exists\ s \leqslant \frac{n}{32\sigma^2\log d}$, s.t.

$$\frac{1}{n}\|A(\hat{w}_{top}(s)-w^*)\|_2^2 < \epsilon$$

(j)

Condition on $\mathscr{E}$. When $\hat{w}_\lambda(s)\neq 0$,

$$\begin{aligned}|e_i| &\leqslant |\hat{w}_\lambda(s)_i - w_i^*|\\ &= |[\hat{w}_\lambda(s)-w^*]_i|\\ &= |z_i'|\\ &\leqslant 2\sigma\sqrt{\log d}\end{aligned}$$

Solution (cont.)

When $\hat{w}_\lambda(s) = 0$,

$$\begin{aligned}
|\hat{w}_\lambda(s)_i| &= |w_i^* + z_i'| \\
&\leqslant \lambda \\
|e_i| &= |w_i^*| \\
&\leqslant |w_i^* + z_i'| + |z_i'| \\
&\leqslant \lambda + 2\sigma\sqrt{\log d}
\end{aligned}$$

$\therefore$

$$\begin{aligned}
|e_i| &= |(\hat{w}_\lambda(s) - w^*)_i| \\
&\leqslant \lambda + 2\sigma\sqrt{\log d}
\end{aligned}$$

$\because$ with probability at least $1 - \frac{1}{d}$,

$$\Pr\left\{\max_{i \in \{1,2,\ldots,d\}} |z_i'|^2 \leq 4\sigma^2 \log d\right\} \leqslant 1 - \frac{1}{d}$$

$\therefore$

$$\begin{aligned}
\|\hat{w}_\lambda(s) - w^*\|_2^2 &= \|e\|_2^2 \\
&\leqslant n(\lambda + 2\sigma\sqrt{\log d})^2 \\
\frac{1}{n}\|\hat{w}_\lambda(s) - w^*\|_2^2 &\leqslant (\lambda + 2\sigma\sqrt{\log d})^2
\end{aligned}$$

Question 4

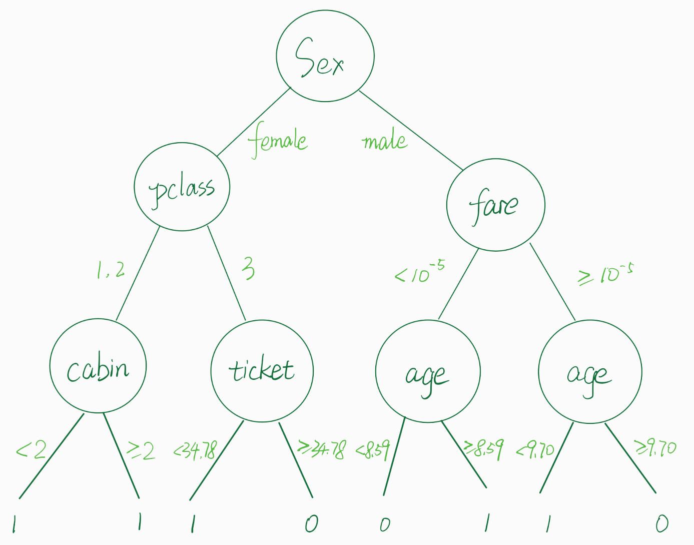
(a)

```
@staticmethod
def entropy(y):
    # TODO implement entropy function
    label = Counter(y)
    Entropy = 0
    n = len(y)
    for i in set(label):
        p = label[i]/n
        Entropy -= p*np.log(p)
    return Entropy

@staticmethod
def information_gain(X, y, thresh):
    # TODO implement information gain function
    info = DecisionTree().entropy(y)
    n = len(y)
    info -= len(y[X<thresh])/n*DecisionTree().entropy(y[X<thresh])
    info -= len(y[X>=thresh])/n*DecisionTree().entropy(y[X>=thresh])
    return info
```

(b)

For continuous variables, I remain the same just like ‘age’, ‘fare’.
For categorical variables, change them into indices. Such variables are ‘pclass’, ‘sex’, ‘sibsp’, ‘parch’, ‘embarked’.
Since ‘ticket’ is less relevant to our problem, we simply delete it.
To ‘cabin’, I encode it into 0 and 1 which means one has a cabin and not respectively.
To deal with the missing value, I replace it with the average value to ‘age’ and remain 0 to ‘fare’.

(c)



(d)

```python
class BaggedTrees(BaseEstimator, ClassifierMixin):

    def __init__(self, params={}, n=200):
        self.params = params
        self.n = n
        self.decision_trees = [
            DecisionTreeClassifier(random_state=i, **self.params)
            for i in range(self.n)]

    def fit(self, X, y):
        # TODO implement function
        m,_ = np.shape(X)
        index = np.random.choice(m,m)
        for i in range(self.n):
            self.decision_trees[i].fit(X[index,:],y[index])

    def predict(self, X):
        # TODO implement function
        result = np.zeros((self.n,np.shape(X)[0]))
        for i in range(self.n):
            result[i,:] = self.decision_trees[i].predict(X)
        return stats.mode(result)[0]
```

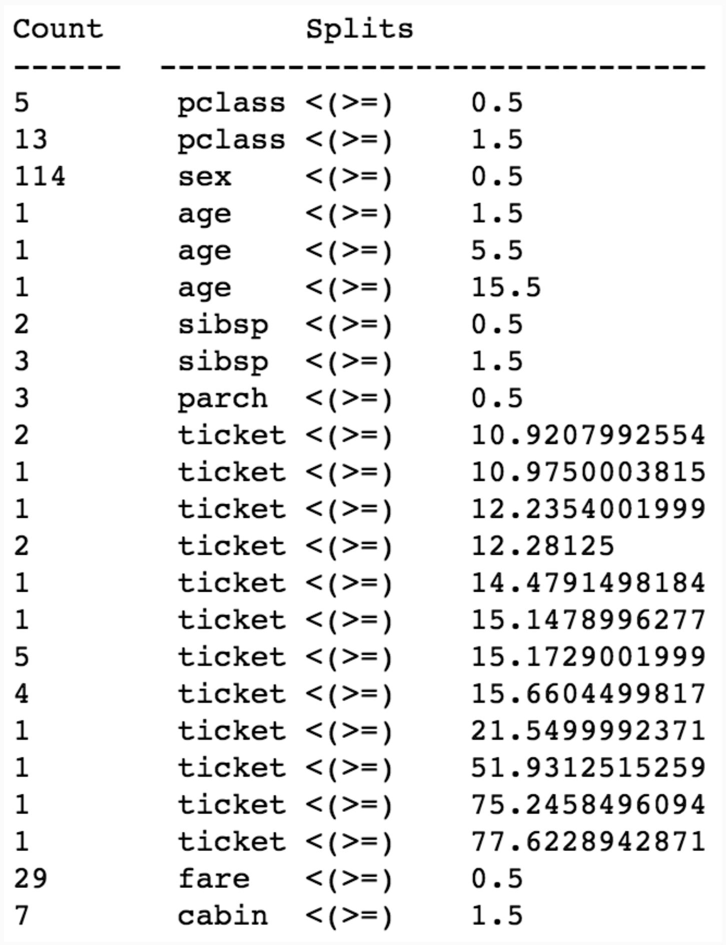
(e)

The most common splits made at the root node of the trees is ‘sex’$\leqslant$0.5 and ‘sex’>0.5.

(f)

```python
class RandomForest(BaggedTrees):
    def __init__(self, params={}, n=200, m=1):
        self.params = params
        self.n = n
        self.m = m
        self.decision_trees = [
            DecisionTreeClassifier(random_state=i, **self.params)
            for i in range(self.n)]
        self.feature = []

    def fit(self, X, y):
        # TODO implement function
        n1,n2 = np.shape(X)
        for i in range(self.n):
            index = np.random.choice(n1,n1)
            fea = np.random.choice(n2,self.m)
            self.decision_trees[i].fit(X[index][:,fea],y[index])
            self.feature.append(fea)

    def predict(self, X):
        # TODO implement function
        result = np.zeros((self.n,np.shape(X)[0]))
        for i in range(self.n):
            result[i,:] = self.decision_trees[i].predict(
              X[:,self.feature[i]])
        return stats.mode(result)[0]
```

(g)



```
Count          Splits
------  ------------------------------
5        pclass <(>=)    0.5
13       pclass <(>=)    1.5
114      sex    <(>=)    0.5
1        age    <(>=)    1.5
1        age    <(>=)    5.5
1        age    <(>=)    15.5
2        sibsp  <(>=)    0.5
3        sibsp  <(>=)    1.5
3        parch  <(>=)    0.5
2        ticket <(>=)    10.9207992554
1        ticket <(>=)    10.9750003815
1        ticket <(>=)    12.2354001999
2        ticket <(>=)    12.28125
1        ticket <(>=)    14.4791498184
1        ticket <(>=)    15.1478996277
5        ticket <(>=)    15.1729001999
4        ticket <(>=)    15.6604499817
1        ticket <(>=)    21.5499992371
1        ticket <(>=)    51.9312515259
1        ticket <(>=)    75.2458496094
1        ticket <(>=)    77.6228942871
29       fare   <(>=)    0.5
7        cabin  <(>=)    1.5
```

(h)

The algorithm increases the weights of the samples that are hard to be classified correctly and increases the weights of the trees that are good at their own training data set.
$a_i < 0$ means that e_i is very large, i.e., the accuracy of this tree is very low. Therefore, the predict score given by this tree will be decrease(so it is negative).

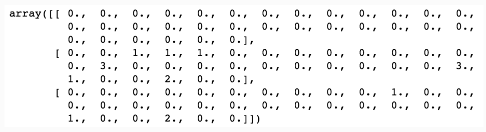
```python
class BoostedRandomForest(RandomForest):
    def fit(self, X, y):
        self.w = np.ones(X.shape[0]) / X.shape[0]  # Weights on data
        self.a = np.zeros(self.n)  # Weights on decision trees
        # TODO implement function
        n1,n2 = np.shape(X)
        for i in range(self.n):
            index = np.random.choice(n1,n1)
            fea = np.random.choice(n2,self.m,replace=False)
            self.decision_trees[i].fit(X[index][:,fea],y[index])
            self.feature.append(fea)
            pred = self.decision_trees[i].predict(X[index][:,fea])
            e = np.mean(pred!=y[index])
```

Solution (cont.)

```
            self.a[i] = 1/2*np.log((1-e)/e)
            for j in range(self.m):
                sign = -1 if pred[j]==y[index[j]] else 1
                self.w[index[j]] *= np.exp(sign*self.a[i])
        return self

    def predict(self, X):
        # TODO implement function
        result = np.zeros((self.n,np.shape(X)[0]))
        for i in range(self.n):
            result[i,:] = self.decision_trees[i].predict(
              X[:,self.feature[i]])
        score = np.sum(result * self.a[:,np.newaxis],axis=0)

        return np.array(score>= np.sum(self.a)/2,dtype=np.int16)
```

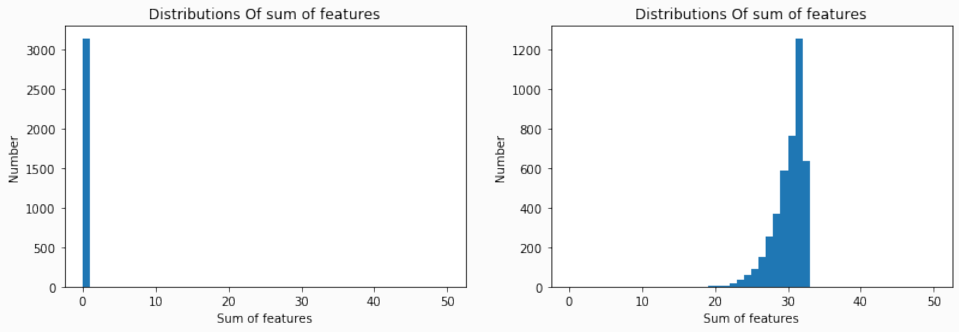
(i)

The sparse data tends to be classified uncorrectly.
First, I print out both the training data classified correctly and uncorrectly respectively. They are like as follows:



```
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  3.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  3.,
         1.,  0.,  0.,  2.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         1.,  0.,  0.,  2.,  0.,  0.]])
```

Then I plot the distribution of these two set :



(j)

Random forests and boosted random forests perform well.
For titanic, I choose $m = 8$ for Random Forests and $m = 7$ for Boosted Random Forests.
For spam, I choose $m = 30$ for Random Forests and $m = 32$ for Boosted Random Forests.

Solution (cont.)

Classifier	Dataset	Training Accuracy			Testing Accuracy		
		Data 1	Data 2	Data 3	Data 1	Data 2	Data 3
A Single Decision Tree	titanic	0.81	0.78	0.82	0.77	0.85	0.77
	spam	0.79	0.79	0.80	0.80	0.80	0.79
Bagged Trees	titanic	0.81	0.78	0.82	0.77	0.85	0.77
	spam	0.80	0.79	0.79	0.79	0.80	0.80
Random Forests	titanic	0.83	0.82	0.87	0.79	0.83	0.75
	spam	0.79	0.78	0.78	0.78	0.77	0.79
Boosted Random Forests	titanic	0.84	0.82	0.87	0.78	0.86	0.78
	spam	0.82	0.82	0.82	0.81	0.81	0.82

(k)

Have submitted.

Question

Question What are the advantages and disadvantages of decision tree?
Solution
Advantages

(1) Simple to understand and interpret. People are able to understand decision tree models after a brief explanation.

(2) Have value even with little hard data. Important insights can be generated based on experts describing a situation (its alternatives, probabilities, and costs) and their preferences for outcomes.

(3) Allow the addition of new possible scenarios.

(4) Help determine worst, best and expected values for different scenarios.

(5) Use a white box model. If a given result is provided by a model.

(6) Can be combined with other decision techniques.

Disadvantages

(1) For data including categorical variables with different number of levels, information gain in decision trees is biased in favor of those attributes with more levels.

(2) Calculations can get very complex, particularly if many values are uncertain and/or if many outcomes are linked.

lasso

November 17, 2017

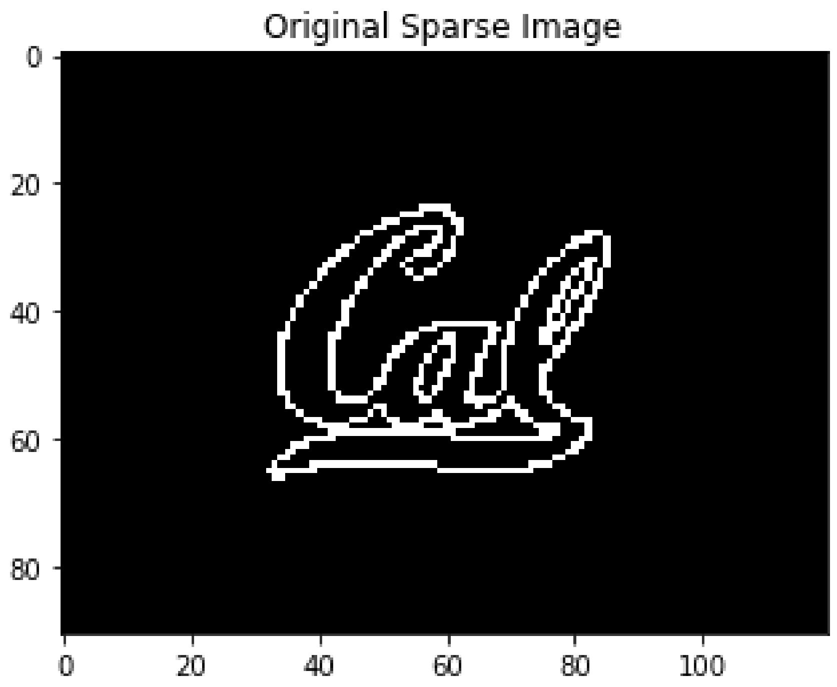
1 Sparse imaging with LASSO

This example generates a sparse signal and tries to recover it using lasso

```
In [1]: from __future__ import print_function
        from __future__ import division
        from sklearn import linear_model
        import matplotlib.pyplot as plt
        import numpy as np
        from scipy import misc
        from IPython import display
        from simulator import *
        %matplotlib inline
```

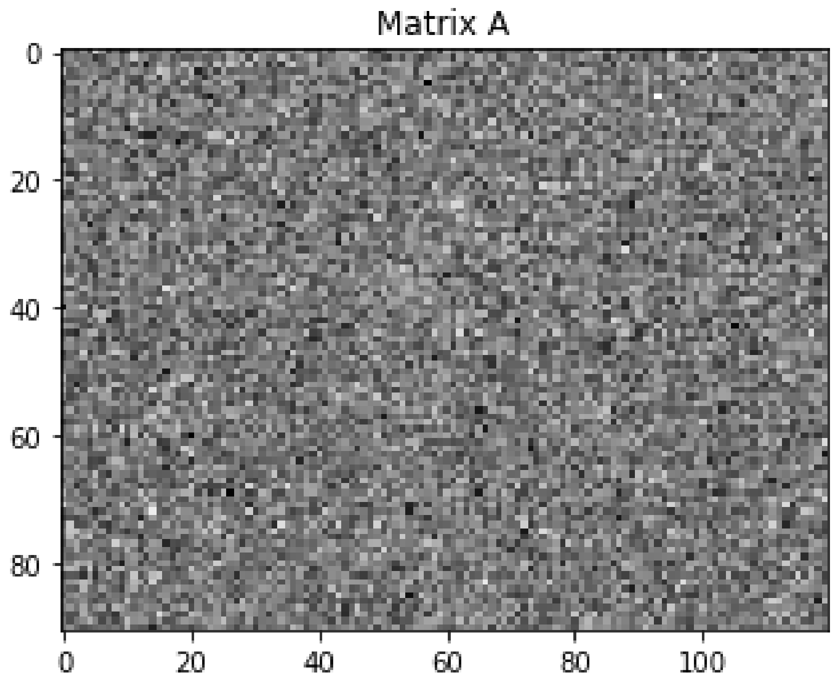
We generate an orthogonal matrix A and compute measurements = Aw+z where w is the vectorized format of the sparse image

```
In [2]: measurements,A,I = simulate()

        # THE SETTINGS FOR THE IMAGE - PLEASE DO NOT CHANGE
        height = 91
        width = 120
        sparsity = 476
        numPixels = len(A[0])

        plt.imshow(I, cmap=plt.cm.gray, interpolation='nearest');
        plt.title('Original Sparse Image')

Out[2]: <matplotlib.text.Text at 0x1152a0320>
```



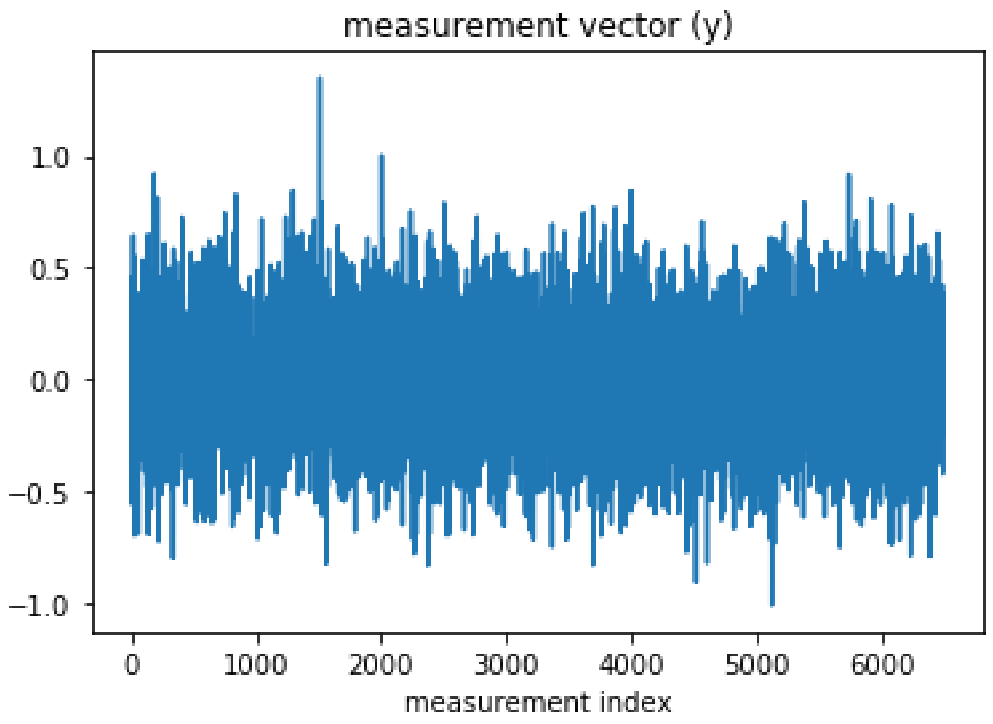
We plot matrix A:

```
In [3]: chosenMaskToDisplay = 0
        M0 = A[chosenMaskToDisplay].reshape((height,width))
        plt.title('Matrix A')
        plt.imshow(M0, cmap=plt.cm.gray, interpolation='nearest');
```



And here is the plot of measurement vector:

```
In [4]: # measurements
        plt.title('measurement vector (y)')
        plt.plot(measurements)
        plt.xlabel('measurement index')
        plt.show()
```



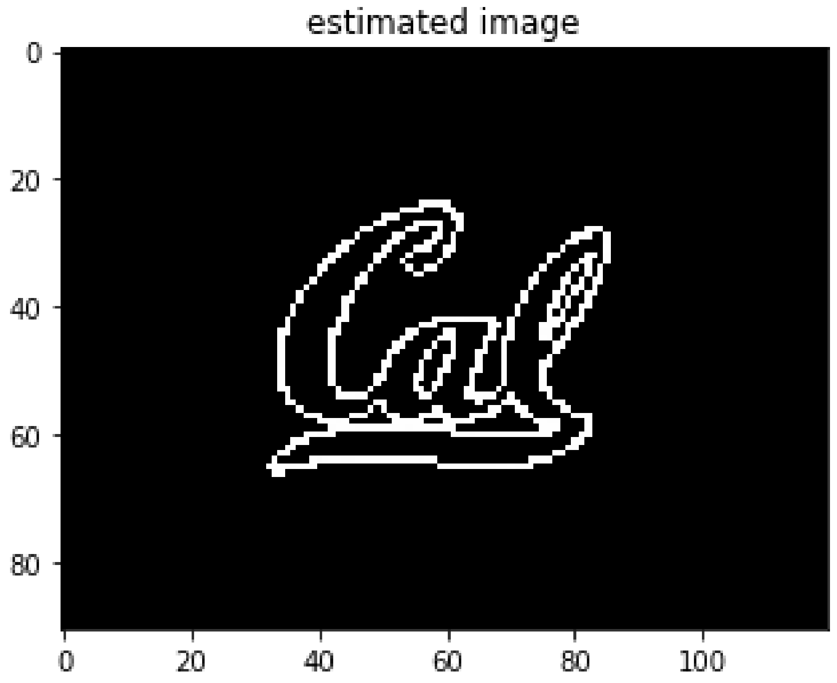
We use lasso to recover the image:

```
In [5]: def LASSO(imDims, measurements, A, a):
            clf = linear_model.Lasso(alpha=a)
            clf.fit(A,measurements)
            Ihat = clf.coef_.reshape(imDims)
            plt.title('estimated image')
            plt.imshow(Ihat, cmap=plt.cm.gray, interpolation='nearest')
            return clf.coef_
```

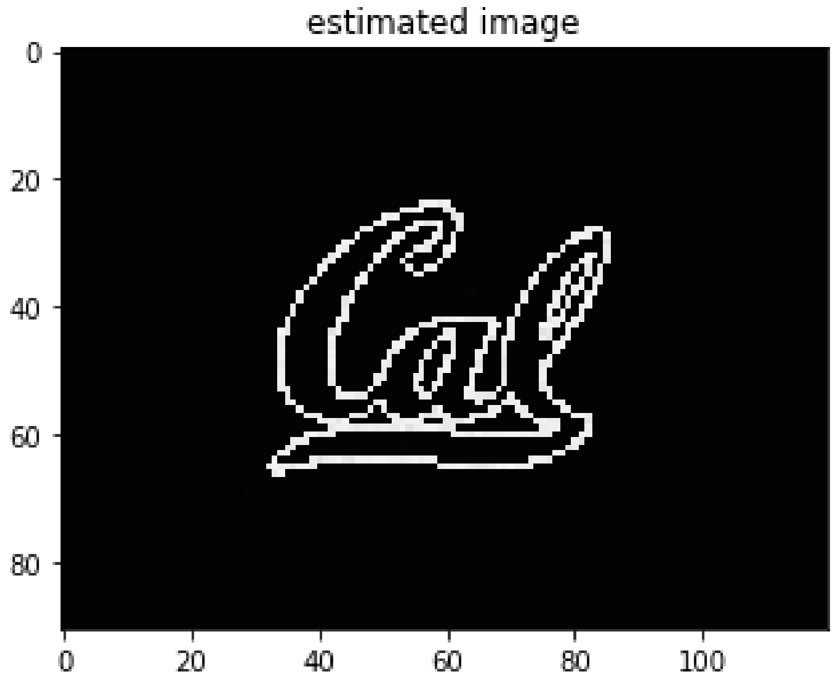
Change the lasso regularization parameter to recover the image and report the value.

```
In [8]: # change the lasso parameter here:
        a = 0.000001
        recovered = LASSO((height,width),measurements,A,a)
        print(recovered)
```

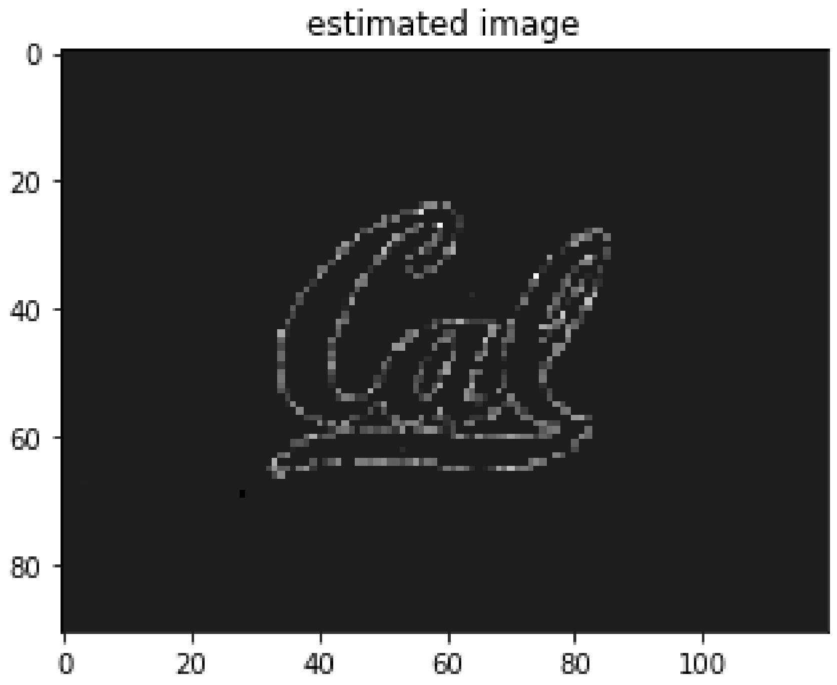
```
[ 0.  0. -0. ..., -0.  0.  0.]
```



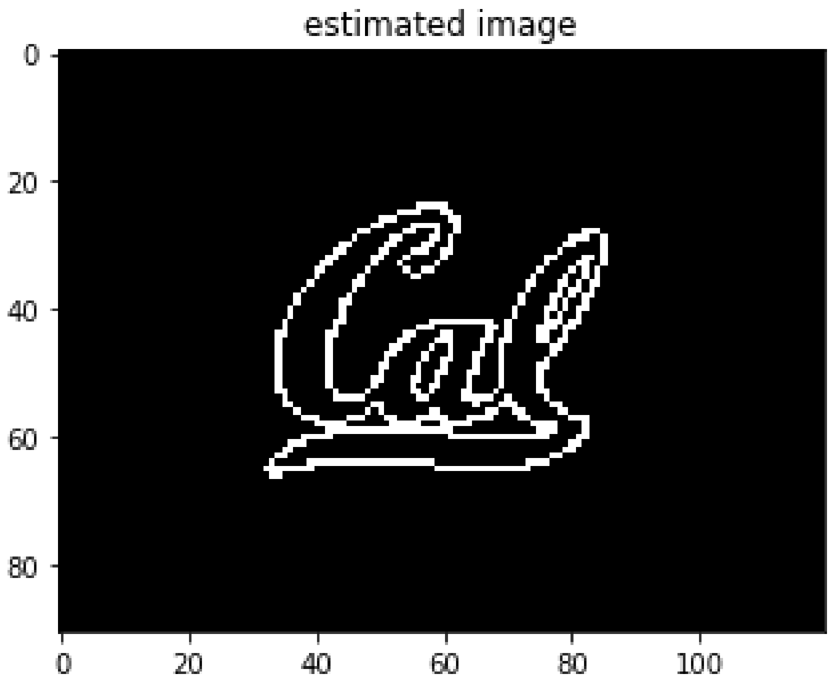
```
In [9]: a = 0.00001
        recovered = LASSO((height,width),measurements,A,a)
```



```
In [10]: a = 0.0001
         recovered = LASSO((height,width),measurements,A,a)
```

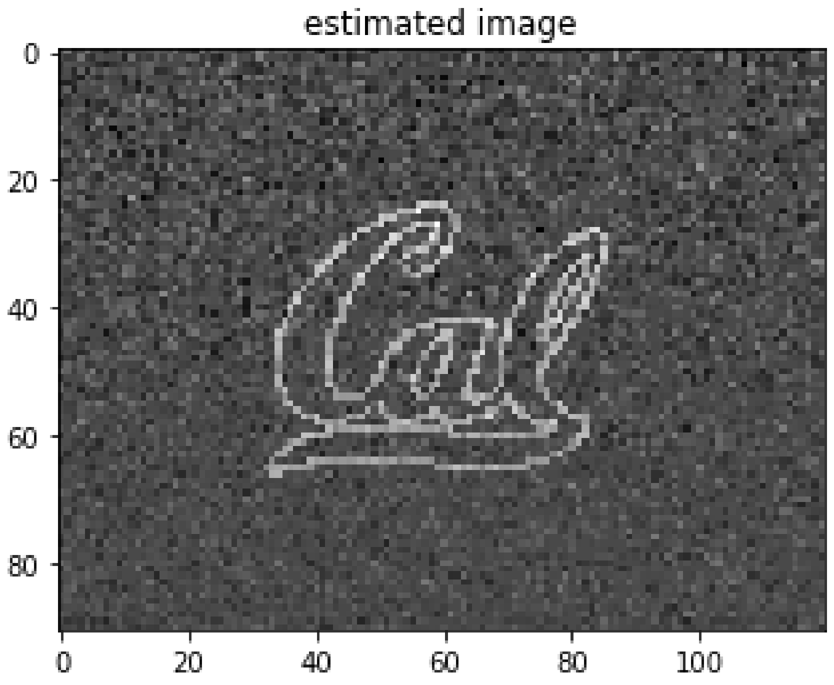


```
In [11]: a = 0.0000001
         recovered = LASSO((height,width),measurements,A,a)
```

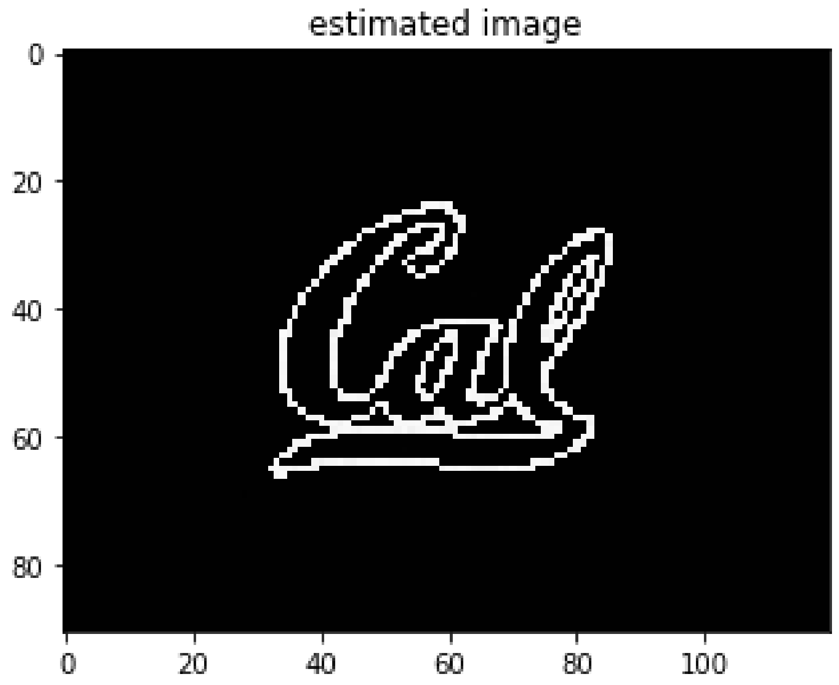


```
In [12]: a = 0.00000001
         recovered = LASSO((height,width),measurements,A,a)
```

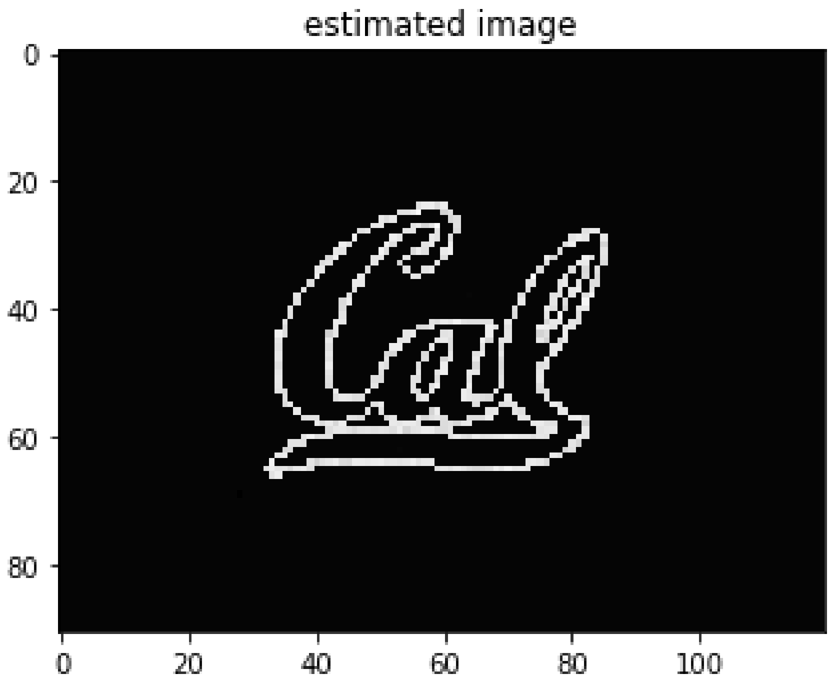
```
/anaconda/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:484: Convergenc
  ConvergenceWarning)
```



```
In [13]: a = 0.000005
         recovered = LASSO((height,width),measurements,A,a)
```



```
In [14]: a = 0.000015
         recovered = LASSO((height,width),measurements,A,a)
```



```
In [ ]:
```

Sparse_Linear_Regression

November 17, 2017

1 Bias and Variance of Sparse Linear Regression

In this notebook, you will explore numerically how sparse vectors change the rate at which we can estimate the underlying model. This corresponds to parts (a), (b), (c) of Homework 12.

First, some setup. We will only be using basic libraries.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
```

The following functions produce the ground truth matrix $A \in \mathbb{R}^{n\times d}$ (denoted by U since it is unitary), as well as the vector $w^* \in \mathbb{R}^d$ and observations $y \in \mathbb{R}^n$. They have been implemented for you, but it is worth going through the code to observe its limitations.

```
In [2]: 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 colu
                      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
```

We now implement the estimators that we will simulate. The least squares estimator has already been implemented for you. You will be implementing the top k and threshold estimators in part (b), but it is fine to skip this for now and compile.

```
In [3]: 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 threshold o
            Output: The estimate what_{T}(lambda), a d X 1 vector that is hard-thresholded (in a
                    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:
            what = wls
            what[np.abs(what)<lmbda] = 0
            ###############
            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 entri
                    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!
    what = wls
    what[np.abs(what)<what[np.argsort(np.abs(what))[-5]]]=0

    ###############
    return what
```

The following helper function that we have implemented for you returns the error of all three estimators as a function *n*, *d*, or *s*, depending on what you specify. Notice that it has the option to generate the true model with sparsity that need not equal the sparsity demanded by the estimators.

Again, this function can be run without implementing the thresh and topk functions, but some of its returned values should then be ignored.

```
In [4]: 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
            averaged over num_iter experiments.

            Input:
            Output: 4 arrays -- range of parameters, errors of LS, topk, and thresh estimator, r
                    functions have not been implemented yet, then these errors are simply the no
            """
            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, true_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, true_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, true_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)

            return arg_range, wls_error, wtopk_error, wthresh_error
```

We are now ready to perform the parts of the question.

2 Part (a)

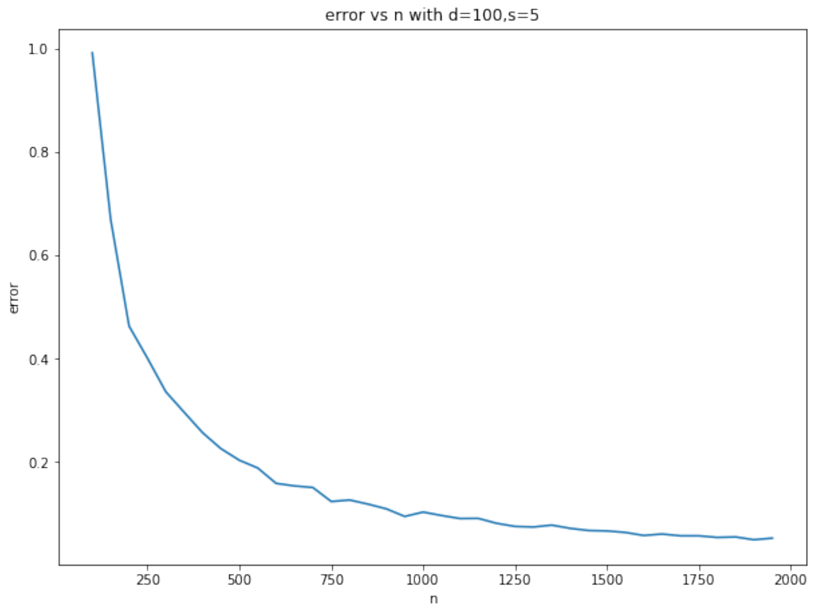
As an example, in the following cell, we run the helper function above to return error values of the OLS estimate for various values of n. You are required to:

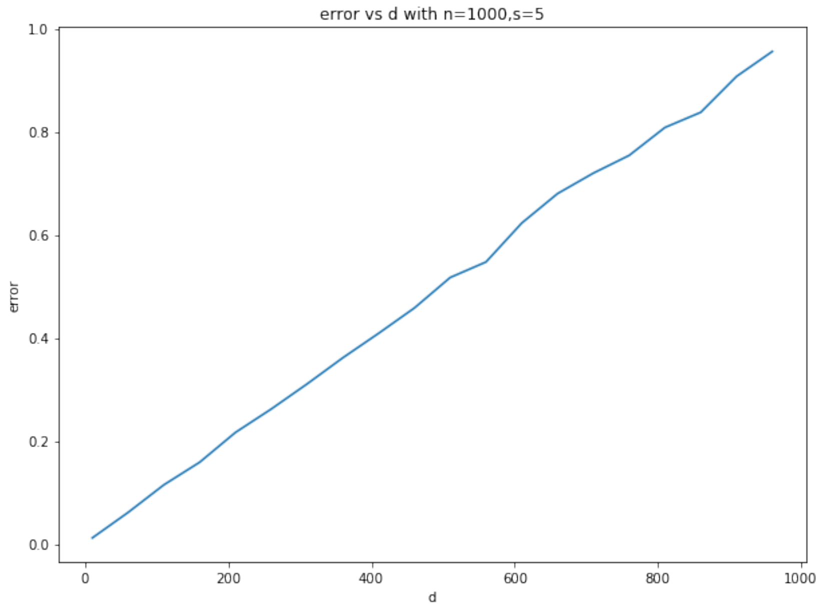
1) Plot the error as a function of n. You may find a log-log plot useful to see the expected bahavior.

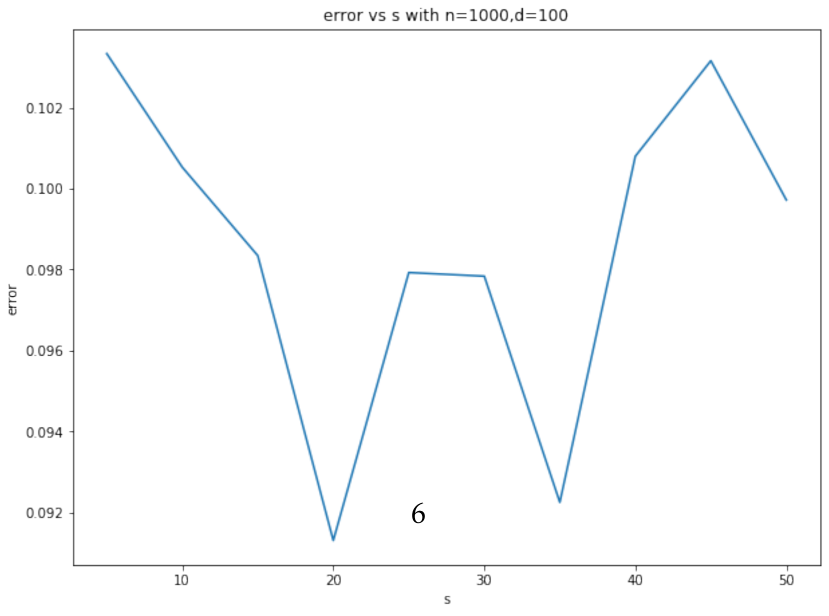
2) Run the helper function to return the error as a function of d and s, and plot your results.

You need to have 3 plots in your answer. Make sure to label axes properly, and to make the plotting visible in general. Feel free to play with the parameters, but ensure that your answer describes your parameter choices. At this point, s_model is True, since we are only interested in the variance of the model.

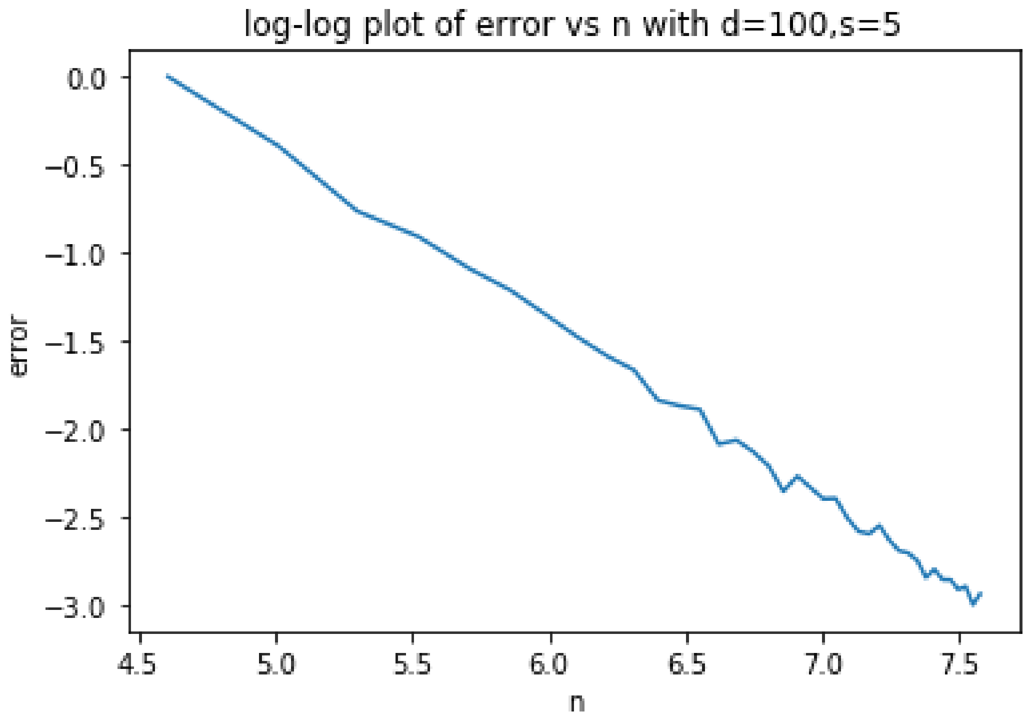
```
In [6]: #nrange contains the range of n used, ls_error the corresponding errors for the OLS esti
        nrange, ls_error, _, _ = error_calc(num_iters=10, param='n', n=1000, d=100, s=5, s_model

        ########
        #TODO: Your code here: call the helper function for d and s, and plot everything
        ########
        #YOUR CODE HERE:
        drange, ls_error_d, _, _ = error_calc(num_iters=10, param='d', n=1000, d=100, s=5, s_mod
        srange, ls_error_s, _, _ = error_calc(num_iters=10, param='s', n=1000, d=100, s=5, s_mod

        plt.figure(figsize=(10,25))
        plt.subplot(3,1,1)
        plt.plot(nrange,ls_error)
        #plt.plot(np.log(nrange),np.log(ls_error))
        plt.title('error vs n with d=100,s=5')
        plt.xlabel('n')
        plt.ylabel('error')
        plt.subplot(3,1,2)
        plt.plot(drange,ls_error_d)
        plt.title('error vs d with n=1000,s=5')
        plt.xlabel('d')
        plt.ylabel('error')
        plt.subplot(3,1,3)
        plt.plot(srange,ls_error_s)
        plt.title('error vs s with n=1000,d=100')
        plt.xlabel('s')
        plt.ylabel('error')

        plt.show()
```


error vs n with d=100,s=5
error
n
1.0
0.8
0.6
0.4
0.2
250
500
750
1000
1250
1500
1750
2000


error vs d with n=1000,s=5
error
d
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000


error vs s with n=1000,d=100
error
s
0.102
0.100
0.098
0.096
0.094
0.092
10
20
30
40
50

```
In [9]: plt.plot(np.log(nrange),np.log(ls_error))
        plt.title('log-log plot of error vs n with d=100,s=5')
        plt.xlabel('n')
        plt.ylabel('error')

Out[9]: <matplotlib.text.Text at 0x1124ee550>
```



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.

2.1 Your answer here

Yes, in the plot, the prediction error seems to have linear relationship with $\frac{1}{n}$ and d respectively. However, the prediction error seems not to have linear relationship with s.

3 Part (b)

Now fill out the functions implementing the sparsity-seeking estimators: thresh, and topk in the above cells. You should be able to test these functions using some straightforward examples.

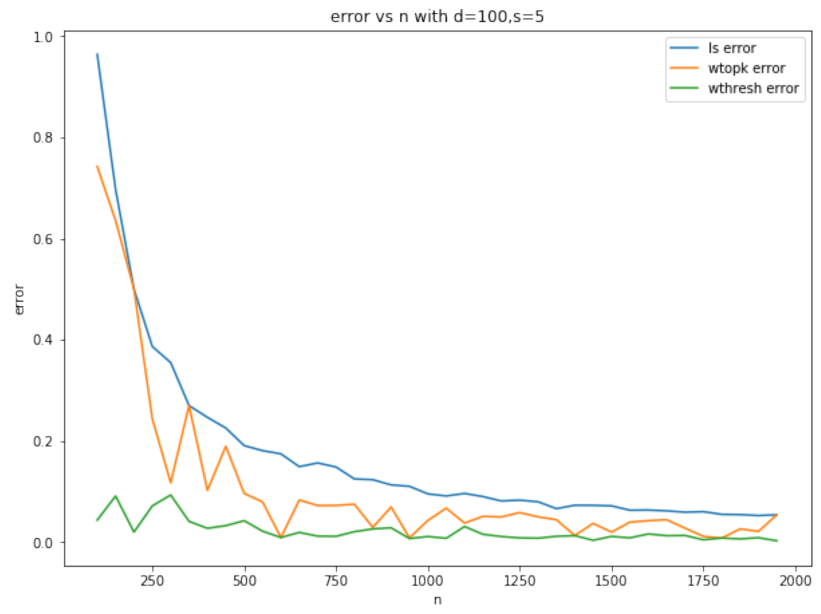
We will now simulate the error of all the estimators, as a function of n, d, and s. An example of this for n is given below. You must:

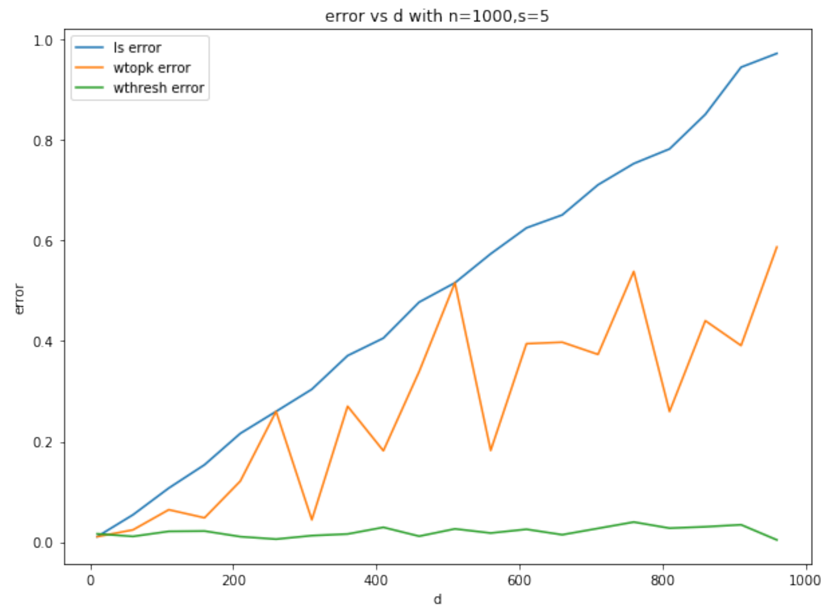
1) Plot the error of all estimators as a function of n.

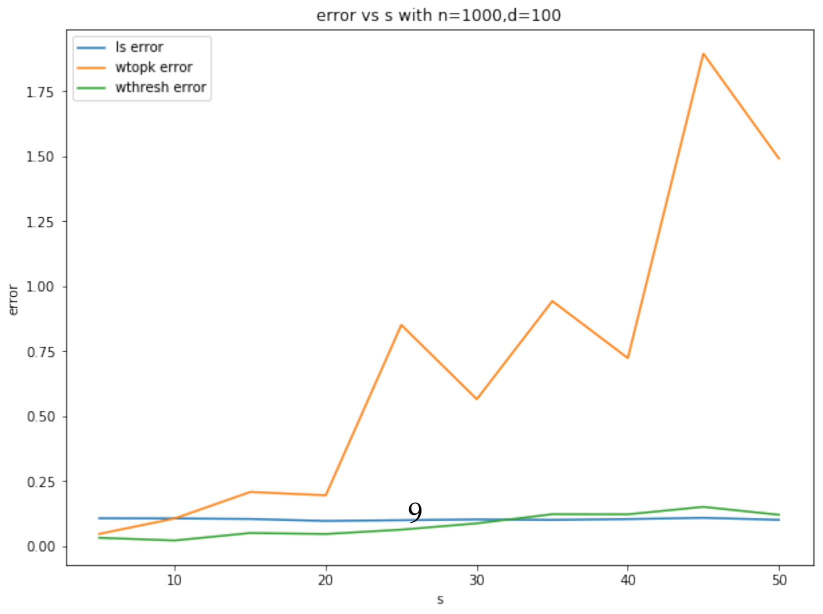
2) Run the helper function to sweep over d and s, and plot the behavior of all three estimators.

You should report 3 plots here once again. Make sure to make them fully readable.

```
In [38]: #TODO: Part (b)
         ##############
         #YOUR CODE HERE:

         nrange, ls_error_n, wtopk_error_n, wthresh_error_n = error_calc(num_iters=10, param='n'
         drange, ls_error_d, wtopk_error_d, wthresh_error_d = error_calc(num_iters=10, param='d'
         srange, ls_error_s, wtopk_error_s, wthresh_error_s = error_calc(num_iters=10, param='s'
         _range = [nrange,drange,srange]
         ls_error = [ls_error_n,ls_error_d,ls_error_s]
         wtopk_error = [wtopk_error_n,wtopk_error_d,wtopk_error_s]
         wthresh_error = [wthresh_error_n,wthresh_error_d,wthresh_error_s]
         title = ['error vs n with d=100,s=5','error vs d with n=1000,s=5','error vs s with n=10
         xlabel = ['n','d','s']
         plt.figure(figsize=(10,25))
         for i in range(3):
             plt.subplot(3,1,i+1)
             plt.plot(_range[i],ls_error[i],label='ls error')
             plt.plot(_range[i],wtopk_error[i],label='wtopk error')
             plt.plot(_range[i],wthresh_error[i],label='wthresh error')
             plt.title(title[i])
             plt.xlabel(xlabel[i])
             plt.ylabel('error')
             plt.legend()
         plt.show()
```


error vs n with d=100,s=5
ls error
wtopk error
wthresh error
error
n


error vs d with n=1000,s=5
ls error
wtopk error
wthresh error
error
d

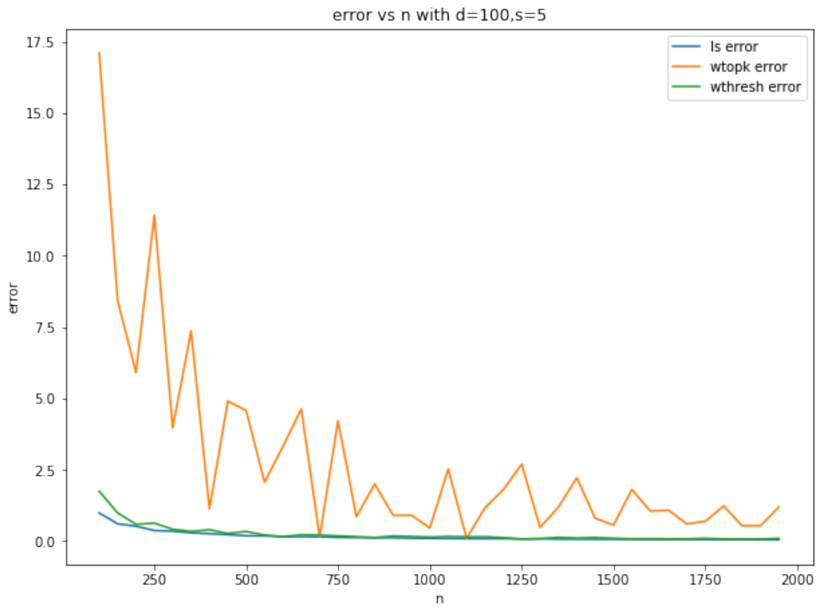

error vs s with n=1000,d=100
ls error
wtopk error
wthresh error
error
s

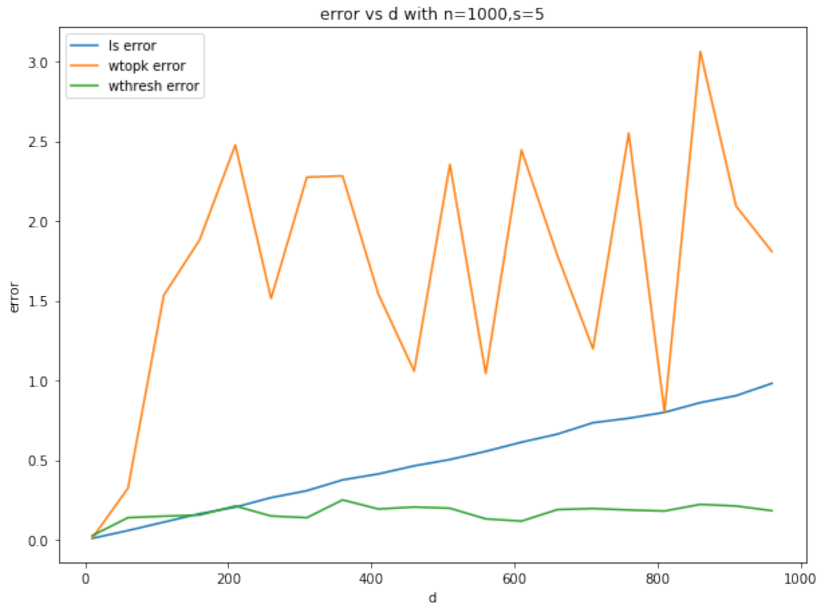
4 Part (c)

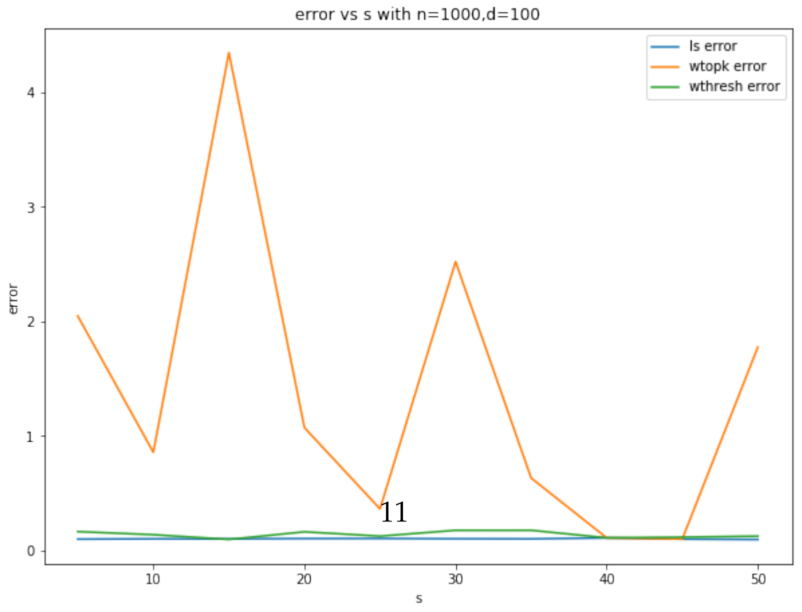
Now, call the helper function with the true sparsity being greater than the sparsity assumed by the top-k estimator. Remember to set s_model to False! Plot the behavior of all three estimators once again, as a function of n, d, s, where s is the assumed sparsity of the top-k model.

You should return 3 plots, and explain what you see in terms of the bias variance tradeoff.

```
In [11]: #TODO: Part (c)
         ##############
         #YOUR CODE HERE:
         nrange, ls_error_n, wtopk_error_n, wthresh_error_n = error_calc(num_iters=10, param='n'
         drange, ls_error_d, wtopk_error_d, wthresh_error_d = error_calc(num_iters=10, param='d'
         srange, ls_error_s, wtopk_error_s, wthresh_error_s = error_calc(num_iters=10, param='s'
         _range = [nrange,drange,srange]
         ls_error = [ls_error_n,ls_error_d,ls_error_s]
         wtopk_error = [wtopk_error_n,wtopk_error_d,wtopk_error_s]
         wthresh_error = [wthresh_error_n,wthresh_error_d,wthresh_error_s]
         title = ['error vs n with d=100,s=5','error vs d with n=1000,s=5','error vs s with n=10
         xlabel = ['n','d','s']
         plt.figure(figsize=(10,25))
         for i in range(3):
             plt.subplot(3,1,i+1)
             plt.plot(_range[i],ls_error[i],label='ls error')
             plt.plot(_range[i],wtopk_error[i],label='wtopk error')
             plt.plot(_range[i],wthresh_error[i],label='wthresh error')
             plt.title(title[i])
             plt.xlabel(xlabel[i])
             plt.ylabel('error')
             plt.legend()
         plt.show()
```


error vs n with d=100,s=5
ls error
wtopk error
wthresh error
error
n
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
250
500
750
1000
1250
1500
1750
2000


error vs d with n=1000,s=5
ls error
wtopk error
wthresh error
error
d
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
200
400
600
800
1000


error vs s with n=1000,d=100
ls error
wtopk error
wthresh error
error
s
0
1
2
3
4
10
20
30
40
50

4.1 Discuss answer to (c) here

Increasing data n helps to reduce the variance just like in plot 1.

Increasing bad features d will increase variance just like in plot 2.

Enforcing sparsity is equivalent to deliberately constraining model complexity. So the variance tends to decrease when $\frac{s}{true\ s}$ decrease.

```
In [ ]:
```

HW 12

November 17, 2017

1 Question 3

```
In [5]: from collections import Counter

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

        eps = 1e-5  # a small number
```

```
In [79]: 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 entropy(y):
                 # TODO implement entropy function
                 label = Counter(y)
                 Entropy = 0
                 n = len(y)
                 for i in set(label):
                     p = label[i]/n
                     Entropy -= p*np.log(p)
                 return Entropy

             @staticmethod
             def information_gain(X, y, thresh):
                 # TODO implement information gain function
```

```
        info = DecisionTree().entropy(y)
        n = len(y)
        info -= len(y[X<thresh])/n*DecisionTree().entropy(y[X<thresh])
        info -= len(y[X>=thresh])/n*DecisionTree().entropy(y[X>=thresh])
        return info

    def split(self, X, y, idx, thresh):
        X0, idx0, X1, idx1 = self.split_test(X, idx=idx, thresh=thresh)
        y0, y1 = y[idx0], y[idx1]
        return X0, y0, X1, y1

    def split_test(self, X, idx, thresh):
        idx0 = np.where(X[:,idx] < thresh)[0]
        idx1 = np.where(X[:,idx] >= thresh)[0]
        X0, X1 = X[idx0, :], X[idx1, :]
        return X0, idx0, X1, idx1

    def fit(self, X, y):
        if self.max_depth > 0:
            # compute entropy gain for all single-dimension splits,
            # thresholding with a linear interpolation of 10 values
            gains = []
            thresh = np.array([np.linspace(np.min(X[:, i]) + eps,
                                           np.max(X[:, i]) - eps, num=10) for i
                               in range(X.shape[1])])
            for i in range(X.shape[1]):
                gains.append([self.information_gain(X[:, i], y, t) for t in
                              thresh[i, :]])

            gains = np.nan_to_num(np.array(gains))
            self.split_idx, thresh_idx = np.unravel_index(np.argmax(gains),
                                                          gains.shape)
            self.thresh = thresh[self.split_idx, thresh_idx]
            X0, y0, X1, y1 = self.split(X, y, idx=self.split_idx,
                                        thresh=self.thresh)
            if X0.size > 0 and X1.size > 0:
                self.left = DecisionTree(max_depth=self.max_depth-1,
                                         feature_labels=self.features)
                self.left.fit(X0, y0)
                self.right = DecisionTree(max_depth=self.max_depth-1,
                                          feature_labels=self.features)
                self.right.fit(X1, y1)
            else:
                self.max_depth = 0
                self.data, self.labels = X, y
                self.pred = stats.mode(y).mode[0]
        else:
            self.data, self.labels = X, y
```

```
                self.pred = stats.mode(y).mode[0]
            return self

        def predict(self, X):
            if self.max_depth == 0:
                return self.pred * np.ones(X.shape[0])
            else:
                X0, idx0, X1, idx1 = self.split_test(X, idx=self.split_idx,
                                                     thresh=self.thresh)
                yhat = np.zeros(X.shape[0])
                yhat[idx0] = self.left.predict(X0)
                yhat[idx1] = self.right.predict(X1)
                return yhat
```

```
In [80]: dataset = "spam"
         params = {
             "max_depth": 5,
             # "random_state": 6,
             "min_samples_leaf": 10,
         }
         N = 100

         if dataset == "titanic":
             # Load titanic data
             path_train = 'titanic_training.csv'
             data = genfromtxt(path_train, delimiter=',', dtype=None)
             path_test = 'titanic_testing_data.csv'
             test_data = genfromtxt(path_test, delimiter=',', dtype=None)
             features = data[0, 1:]  # features = all columns except survived
             y = data[1:, 0]  # label = survived
             class_names = ["Died", "Survived"]

             # TODO implement preprocessing of Titanic dataset
             X, Z = None, None
         elif 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",
                         "pleased", "record", "out", "semicolon", "dollar", "sharp",
                         "exclamation", "parenthesis", "square_bracket", "ampersand"]
             assert len(features) == 32

             # Load spam data
             path_train = 'spam_data.mat'
             #path_test = 'datasets/spam_data/spam_test_labels.txt'
             data = scipy.io.loadmat(path_train)
             X = data['training_data']
```

```
            y = np.squeeze(data['training_labels'])
            #Z = data['test_data']
            class_names = ["Ham", "Spam"]
        else:
            raise NotImplementedError("Dataset %s not handled" % dataset)

        print("Features", features)
        print("Train size", X.shape)

        print("\n\nPart 0: constant classifier")
        print("Accuracy", 1 - np.sum(y) / y.size)

        # Basic decision tree
        print("\n\nPart (a-b): simplified decision tree")
        dt = DecisionTree(max_depth=3, feature_labels=features)
        dt.fit(X, y)
        pred = dt.predict(X)
        print("Accuracy", 1 - np.mean(pred!=y))
        print("Predictions", pred[:100])
```

```
Features ['pain', 'private', 'bank', 'money', 'drug', 'spam', 'prescription', 'creative', 'heigh
Train size (5172, 32)


Part 0: constant classifier
Accuracy 0.709976798144


Part (a-b): simplified decision tree
Accuracy 0.794856921887
Predictions [ 1.  1.  0.  0.  0.  0.  1.  1.  1.  1.  0.  1.  1.  1.  1.  1.  1.  0.
  0.  1.  0.  1.  0.  1.  1.  1.  1.  0.  0.  0.  1.  0.  1.  0.  0.  0.
  1.  1.  1.  1.  1.  1.  1.  1.  0.  1.  0.  0.  0.  1.  1.  1.  0.  0.
  1.  0.  1.  0.  0.  1.  1.  1.  1.  0.  0.  1.  1.  1.  0.  0.  1.  0.
  0.  1.  1.  0.  0.  0.  0.  0.  1.  1.  0.  1.  0.  1.  1.  1.  0.  1.
  0.  0.  0.  1.  1.  0.  1.  1.  0.  1.]
```

1.1 (b)

```
In [83]: params = {
             "max_depth": 5,
             # "random_state": 6,
             "min_samples_leaf": 10,
         }
         N = 100
         # Load titanic data
         path_train = 'titanic_training.csv'
         data = genfromtxt(path_train, delimiter=',', dtype=None)
         path_test = 'titanic_testing_data.csv'
         test_data = genfromtxt(path_test, delimiter=',', dtype=None)
         features = data[0, 1:]  # features = all columns except survived
         class_names = ["Died", "Survived"]
         print(features)

[b'pclass' b'sex' b'age' b'sibsp' b'parch' b'ticket' b'fare' b'cabin'
 b'embarked']


In [84]: data[706,:]
         data = np.delete(data,706,0)
         data[706,:]

Out[84]: array([b'0', b'3', b'male', b'', b'0', b'0', b'376563', b'8.05', b'', b'S'],
               dtype='|S18')

In [85]: y = data[1:, 0]  # label = survived
         X = np.zeros_like(data[1:,2:],dtype=np.float32)
         Z = np.zeros_like(test_data[1:,1:],dtype=np.float32)

In [86]: from sklearn import preprocessing

In [87]: pclass = preprocessing.LabelEncoder()
         pclass.fit(data[1:,1])
         X[:,0] = pclass.transform(data[1:,1])
         print(pclass.classes_)
         X

[b'1' b'2' b'3']


Out[87]: array([[ 2.,  0.,  0., ...,  0.,  0.,  0.],
                [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                [ 1.,  0.,  0., ...,  0.,  0.,  0.],
                ...,
                [ 1.,  0.,  0., ...,  0.,  0.,  0.],
                [ 2.,  0.,  0., ...,  0.,  0.,  0.],
                [ 1.,  0.,  0., ...,  0.,  0.,  0.]], dtype=float32)

In [88]: sex = preprocessing.LabelEncoder()
         sex.fit(data[1:,2])
         X[:,1] = sex.transform(data[1:,2])
         print(sex.classes_)
         X

[b'female' b'male']
```

```
Out[88]: array([[ 2.,  1.,  0., ...,  0.,  0.,  0.],
                [ 0.,  1.,  0., ...,  0.,  0.,  0.],
                [ 1.,  1.,  0., ...,  0.,  0.,  0.],
                ...,
                [ 1.,  1.,  0., ...,  0.,  0.,  0.],
                [ 2.,  0.,  0., ...,  0.,  0.,  0.],
                [ 1.,  1.,  0., ...,  0.,  0.,  0.]], dtype=float32)
```

```
In [89]: age = lambda x:[0 if i==b'' else float(i) for i in x]
         X[:,2] = age(data[1:,3])
         average_age = np.mean(X[:,2])
         X[X[:,2]==0,2] = average_age
         X
```

```
Out[89]: array([[  2.        ,   1.        ,  24.3529377, ...,   0.        ,
                   0.        ,   0.        ],
                [  0.        ,   1.        ,  22.       , ...,   0.        ,
                   0.        ,   0.        ],
                [  1.        ,   1.        ,  23.       , ...,   0.        ,
                   0.        ,   0.        ],
                ...,
                [  1.        ,   1.        ,  63.       , ...,   0.        ,
                   0.        ,   0.        ],
                [  2.        ,   0.        ,  41.       , ...,   0.        ,
                   0.        ,   0.        ],
                [  1.        ,   1.        ,  34.       , ...,   0.        ,
                   0.        ,   0.        ]], dtype=float32)
```

```
In [90]: sibsp = preprocessing.LabelEncoder()
         sibsp.fit(data[1:,4])
         X[:,3] = sibsp.transform(data[1:,4])
         print(sibsp.classes_)
         X
```

```
[b'0' b'1' b'2' b'3' b'4' b'5' b'8']
```

```
Out[90]: array([[  2.        ,   1.        ,  24.3529377, ...,   0.        ,
                   0.        ,   0.        ],
                [  0.        ,   1.        ,  22.       , ...,   0.        ,
                   0.        ,   0.        ],
                [  1.        ,   1.        ,  23.       , ...,   0.        ,
                   0.        ,   0.        ],
                ...,
                [  1.        ,   1.        ,  63.       , ...,   0.        ,
                   0.        ,   0.        ],
                [  2.        ,   0.        ,  41.       , ...,   0.        ,
                   0.        ,   0.        ],
                [  1.        ,   1.        ,  34.       , ...,   0.        ,
                   0.        ,   0.        ]], dtype=float32)
```

```
In [91]: parch = preprocessing.LabelEncoder()
         parch.fit(data[1:,5])
         X[:,4] = parch.transform(data[1:,5])
         print(parch.classes_)
         X

[b'0' b'1' b'2' b'3' b'4' b'5' b'6' b'9']


Out[91]: array([[  2.        ,   1.        ,  24.3529377, ...,   0.        ,
                   0.        ,   0.        ],
                [  0.        ,   1.        ,  22.       , ...,   0.        ,
                   0.        ,   0.        ],
                [  1.        ,   1.        ,  23.       , ...,   0.        ,
                   0.        ,   0.        ],
                ...,
                [  1.        ,   1.        ,  63.       , ...,   0.        ,
                   0.        ,   0.        ],
                [  2.        ,   0.        ,  41.       , ...,   0.        ,
                   0.        ,   0.        ],
                [  1.        ,   1.        ,  34.       , ...,   0.        ,
                   0.        ,   0.        ]], dtype=float32)

In [92]: fare = lambda x:[0 if i==b'' else float(i) for i in x]
         X[:,5] = fare(data[1:,7])
         X

Out[92]: array([[   2.        ,    1.        ,   24.3529377 , ...,    8.05000019,
                    0.        ,    0.        ],
                [   0.        ,    1.        ,   22.        , ...,  135.63330078,
                    0.        ,    0.        ],
                [   1.        ,    1.        ,   23.        , ...,   15.04580021,
                    0.        ,    0.        ],
                ...,
                [   1.        ,    1.        ,   63.        , ...,   26.        ,
                    0.        ,    0.        ],
                [   2.        ,    0.        ,   41.        , ...,   39.6875    ,
                    0.        ,    0.        ],
                [   1.        ,    1.        ,   34.        , ...,   26.        ,
                    0.        ,    0.        ]], dtype=float32)

In [93]: cabin = lambda x:[0 if i==b'' else 1 for i in x]
         X[:,6] = cabin(data[1:,8])
         X

Out[93]: array([[   2.        ,    1.        ,   24.3529377 , ...,    8.05000019,
                    0.        ,    0.        ],
                [   0.        ,    1.        ,   22.        , ...,  135.63330078,
                    0.        ,    0.        ],
                [   1.        ,    1.        ,   23.        , ...,   15.04580021,
                    0.        ,    0.        ],
                ...,
                [   1.        ,    1.        ,   63.        , ...,   26.        ,
                    0.        ,    0.        ],
                [   2.        ,    0.        ,   41.        , ...,   39.6875    ,
                    0.        ,    0.        ],
                [   1.        ,    1.        ,   34.        , ...,   26.        ,
                    0.        ,    0.        ]], dtype=float32)
```

```
In [94]: embarked = preprocessing.LabelEncoder()
         embarked.fit(data[1:,9])
         X[:,7] = embarked.transform(data[1:,9])
         print(embarked.classes_)
         X
```

```
[b'' b'C' b'Q' b'S']
```

```
Out[94]: array([[   2.        ,    1.        ,   24.3529377 , ...,    8.05000019,
                    0.        ,    3.        ],
                [   0.        ,    1.        ,   22.        , ...,  135.63330078,
                    0.        ,    1.        ],
                [   1.        ,    1.        ,   23.        , ...,   15.04580021,
                    0.        ,    1.        ],
                ...,
                [   1.        ,    1.        ,   63.        , ...,   26.        ,
                    0.        ,    3.        ],
                [   2.        ,    0.        ,   41.        , ...,   39.6875    ,
                    0.        ,    3.        ],
                [   1.        ,    1.        ,   34.        , ...,   26.        ,
                    0.        ,    3.        ]], dtype=float32)
```

```
In [95]: Z[:,0] = pclass.transform(test_data[1:,0])
         Z[:,1] = sex.transform(test_data[1:,1])
         Z[:,2] = age(test_data[1:,2])
         Z[:,3] = sibsp.transform(test_data[1:,3])
         Z[:,4] = parch.transform(test_data[1:,4])
         Z[:,5] = fare(test_data[1:,6])
         Z[:,6] = cabin(test_data[1:,7])
         Z[:,7] = embarked.transform(test_data[1:,8])
         Z
```

```
Out[95]: array([[  0.        ,   0.        ,  24.        , ...,  69.30000305,
                   1.        ,   1.        ],
                [  0.        ,   0.        ,  44.        , ...,  57.97919846,
                   1.        ,   1.        ],
                [  2.        ,   1.        ,   1.        , ...,  46.90000153,
                   0.        ,   3.        ],
       ...,
       [ 0.        ,  1.        , 42.        , ...,  26.28750038,
         1.        ,  3.        ],
       [ 2.        ,  1.        ,  0.        , ...,   7.75      ,
         0.        ,  2.        ],
       [ 1.        ,  1.        , 35.        , ...,  12.35000038,
         0.        ,  2.        ]], dtype=float32)
```

```
In [96]: label = preprocessing.LabelEncoder()
         label.fit(y)
         y = label.transform(y)
         y
```

```
Out[96]: array([0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
                1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
                0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1,
                0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1,
                0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0,
                0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1,
                1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
                1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1,
                1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0,
                0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0,
                1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1,
                1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
                0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0,
                1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
                1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1,
                1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 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, 1, 1, 0, 0, 0, 0, 0, 0,
                1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0,
                1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
                1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0,
                0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
                0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0,
                0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1,
                1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0,
                0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0,
                1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
                0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1,
                0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
                0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
                0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
                1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
                0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
                0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0,
                1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1,
                0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
                1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
                0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
                1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1,
                0, 0, 1, 1, 1, 0, 1, 0, 0, 0])
```

```
In [97]: with open('zdata','wb') as f:
             np.save(f,[X,Z,y])
```

1.2 (c)

```
In [403]: with open('zdata','rb') as f:
              zdata = np.load(f)
          X = zdata[0]
          Z = zdata[1]
          y = zdata[2]
```

```
In [404]: print("Features", features)
          print("Train size", X.shape)

          print("\n\nPart 0: constant classifier")
          print("Accuracy", 1 - np.sum(y) / y.size)

          # Basic decision tree
          print("\n\nPart (a-b): simplified decision tree")
          dt = DecisionTree(max_depth=3, feature_labels=features)
          dt.fit(X, y)
          pred = dt.predict(X)
          print("Accuracy", 1 - np.mean(pred!=y))
          #print("Predictions", pred[:100])
```

```
Features ['pain', 'private', 'bank', 'money', 'drug', 'spam', 'prescription', 'creative', 'heigh
Train size (999, 8)


Part 0: constant classifier
Accuracy 0.613613613614


Part (a-b): simplified decision tree
Accuracy 0.802802802803
```

```
In [405]: t = dt
          tree_list = [t]
          zfeatures = ['pclass','sex','age','sibsp','parch','ticket','fare','cabin','embarked']
          while(tree_list):
              temp = []
              for i in tree_list:
                  if i.pred!=None:
                      print(i.pred,end=' ')
                  else:
                      print(zfeatures[i.split_idx],i.thresh,end=' ')
                  if i.left:
                      temp.append(i.left)
                  if i.right:
                      temp.append(i.right)
              tree_list = temp
              print()
```

```
sex 1e-05
pclass 1.11111 fare 1e-05
cabin 2.33332777778 ticket 34.7722245837 age 8.59263000492 age 9.70374111603
1 1 1 0 0 0 1 0
```

```
In [256]: dataset = "spam"
          params = {
              "max_depth": 5,
              # "random_state": 6,
              "min_samples_leaf": 10,
          }
          N = 100

          features = ["pain", "private", "bank", "money", "drug", "spam",
                      "prescription", "creative", "height", "featured", "differ",
                      "width", "other", "energy", "business", "message",
                      "volumes", "revision", "path", "meter", "memo", "planning",
                      "pleased", "record", "out", "semicolon", "dollar", "sharp",
                      "exclamation", "parenthesis", "square_bracket", "ampersand"]
          assert len(features) == 32

          # Load spam data
          path_train = 'spam_data.mat'
          #path_test = 'datasets/spam_data/spam_test_labels.txt'
          data = scipy.io.loadmat(path_train)
          X = data['training_data']
          y = np.squeeze(data['training_labels'])
          #Z = data['test_data']
          class_names = ["Ham", "Spam"]


          print("Features", features)
```

```
        print("Train size", X.shape)

        print("\n\nPart 0: constant classifier")
        print("Accuracy", 1 - np.sum(y) / y.size)

        # Basic decision tree
        print("\n\nPart (a-b): simplified decision tree")
        dt = DecisionTree(max_depth=3, feature_labels=features)
        dt.fit(X, y)
        pred = dt.predict(X)
        print("Accuracy", 1 - np.mean(pred!=y))
        print("Predictions", pred[:100])
```

```
Features ['pain', 'private', 'bank', 'money', 'drug', 'spam', 'prescription', 'creative', 'heigh
Train size (5172, 32)


Part 0: constant classifier
Accuracy 0.709976798144


Part (a-b): simplified decision tree
Accuracy 0.794856921887
Predictions [ 1.  1.  0.  0.  0.  0.  1.  1.  1.  1.  0.  1.  1.  1.  1.  1.  1.  0.
  0.  1.  0.  1.  0.  1.  1.  1.  1.  0.  0.  0.  1.  0.  1.  0.  0.  0.
  1.  1.  1.  1.  1.  1.  1.  1.  0.  1.  0.  0.  0.  1.  1.  1.  0.  0.
  1.  0.  1.  0.  0.  1.  1.  1.  1.  0.  0.  1.  1.  1.  0.  0.  1.  0.
  0.  1.  1.  0.  0.  0.  0.  0.  1.  1.  0.  1.  0.  1.  1.  1.  0.  1.
  0.  0.  0.  1.  1.  0.  1.  1.  0.  1.]
```

1.3 (d)

```
In [168]: class BaggedTrees(BaseEstimator, ClassifierMixin):

              def __init__(self, params={}, n=200):
                  self.params = params
                  self.n = n
                  self.decision_trees = [
                      DecisionTreeClassifier(random_state=i, **self.params) for i in
                      range(self.n)]

              def fit(self, X, y):
                  # TODO implement function
                  m,_ = np.shape(X)
                  for i in range(self.n):
                      index = np.random.choice(m,m)
                      self.decision_trees[i].fit(X[index,:],y[index])
```

```
    def predict(self, X):
        # TODO implement function
        result = np.zeros((self.n,np.shape(X)[0]))
        for i in range(self.n):
            result[i,:] = self.decision_trees[i].predict(X)
        return stats.mode(result)[0]
```

1.4 (e)

```
In [171]: with open('zdata','rb') as f:
              zdata = np.load(f)
          X = zdata[0]
          Z = zdata[1]
          y = zdata[2]
          print("Features", features)
          print("Train size", X.shape)

          print("\n\nPart 0: constant classifier")
          print("Accuracy", 1 - np.sum(y) / y.size)

          # Basic decision tree
          print("\n\nPart (d-e): Bagged decision tree")
          np.random.seed(1)
          bt = BaggedTrees(params)
          bt.fit(X, y)
          pred = bt.predict(X)
          print("Accuracy", 1 - np.mean(pred!=y))
          #print("Predictions", pred[:100])
```

```
Features [b'pclass' b'sex' b'age' b'sibsp' b'parch' b'ticket' b'fare' b'cabin'
 b'embarked']
Train size (999, 8)


Part 0: constant classifier
Accuracy 0.613613613614


Part (d-e): Bagged decision tree
Accuracy 0.831831831832
```

```
In [205]: common_split = []
          for i in range(bt.n):
              common_split.append([bt.decision_trees[i].tree_.feature[0],bt.decision_trees[i].tr
              #print(bt.decision_trees[i].tree_.threshold)
              #print(bt.decision_trees[i].tree_.feature)
          np.unique(common_split,axis=0)
```

```
Out[205]: array([[ 1. ,  0.5]])

In [257]: dataset = "spam"
          params = {
              "max_depth": 5,
              # "random_state": 6,
              "min_samples_leaf": 10,
          }
          N = 100

          features = ["pain", "private", "bank", "money", "drug", "spam",
                      "prescription", "creative", "height", "featured", "differ",
                      "width", "other", "energy", "business", "message",
                      "volumes", "revision", "path", "meter", "memo", "planning",
                      "pleased", "record", "out", "semicolon", "dollar", "sharp",
                      "exclamation", "parenthesis", "square_bracket", "ampersand"]
          assert len(features) == 32

          # Load spam data
          path_train = 'spam_data.mat'
          #path_test = 'datasets/spam_data/spam_test_labels.txt'
          data = scipy.io.loadmat(path_train)
          X = data['training_data']
          y = np.squeeze(data['training_labels'])
          #Z = data['test_data']
          class_names = ["Ham", "Spam"]


          print("Features", features)
          print("Train size", X.shape)

          print("\n\nPart 0: constant classifier")
          print("Accuracy", 1 - np.sum(y) / y.size)

          # Basic decision tree
          print("\n\nPart (a-b): simplified decision tree")
          bt = DecisionTree(max_depth=3, feature_labels=features)
          bt.fit(X, y)
          pred = bt.predict(X)
          print("Accuracy", 1 - np.mean(pred!=y))
          print("Predictions", pred[:100])

Features ['pain', 'private', 'bank', 'money', 'drug', 'spam', 'prescription', 'creative', 'heigh
Train size (5172, 32)


Part 0: constant classifier
Accuracy 0.709976798144
```

```
Part (a-b): simplified decision tree
Accuracy 0.794856921887
Predictions [ 1.  1.  0.  0.  0.  0.  1.  1.  1.  1.  0.  1.  1.  1.  1.  1.  1.  0.
  0.  1.  0.  1.  0.  1.  1.  1.  1.  0.  0.  0.  1.  0.  1.  0.  0.  0.
  1.  1.  1.  1.  1.  1.  1.  1.  0.  1.  0.  0.  0.  1.  1.  1.  0.  0.
  1.  0.  1.  0.  0.  1.  1.  1.  1.  0.  0.  1.  1.  1.  0.  0.  1.  0.
  0.  1.  1.  0.  0.  0.  0.  0.  1.  1.  0.  1.  0.  1.  1.  1.  0.  1.
  0.  0.  0.  1.  1.  0.  1.  1.  0.  1.]
```

1.5 (f)

```
In [280]: class RandomForest(BaggedTrees):
              def __init__(self, params={}, n=200, m=1):
                  self.params = params
                  self.n = n
                  self.m = m
                  self.decision_trees = [
                      DecisionTreeClassifier(random_state=i, **self.params) for i in
                      range(self.n)]
                  self.feature = []

              def fit(self, X, y):
                  # TODO implement function
                  n1,n2 = np.shape(X)
                  for i in range(self.n):
                      index = np.random.choice(n1,n1)
                      num = np.random.choice(self.m,1)+1
                      fea = np.random.choice(n2,num,replace=False)
                      self.decision_trees[i].fit(X[index][:,fea],y[index])
                      self.feature.append(fea)

              def predict(self, X):
                  # TODO implement function
                  result = np.zeros((self.n,np.shape(X)[0]))
                  for i in range(self.n):
                      result[i,:] = self.decision_trees[i].predict(X[:,self.feature[i]])
                  return stats.mode(result)[0]
```

1.6 (g)

```
In [282]: with open('zdata','rb') as f:
              zdata = np.load(f)
          X = zdata[0]
          Z = zdata[1]
          y = zdata[2]
```

```
        print("Features", features)
        print("Train size", X.shape)

        print("\n\nPart 0: constant classifier")
        print("Accuracy", 1 - np.sum(y) / y.size)

        # Basic decision tree
        print("\n\nPart (f-g): Bagged Random Forest")
        m = [1,2,3,4,5,6,7,8]
        rf = [RandomForest(params,m=m[i]) for i in range(len(m))]
        for i in range(len(m)):
            np.random.seed(1)
            rf[i].fit(X, y)
            pred = rf[i].predict(X)
            print("Random Forese with feature subset size m=",m[i])
            print("Accuracy", 1 - np.mean(pred!=y))
        #print("Predictions", pred[:100])
```

```
Features ['pain', 'private', 'bank', 'money', 'drug', 'spam', 'prescription', 'creative', 'heigh
Train size (999, 8)


Part 0: constant classifier
Accuracy 0.613613613614


Part (f-g): Bagged Random Forest
Random Forese with feature subset size m= 1
Accuracy 0.704704704705
Random Forese with feature subset size m= 2
Accuracy 0.727727727728
Random Forese with feature subset size m= 3
Accuracy 0.75975975976
Random Forese with feature subset size m= 4
Accuracy 0.770770770771
Random Forese with feature subset size m= 5
Accuracy 0.77977977978
Random Forese with feature subset size m= 6
Accuracy 0.804804804805
Random Forese with feature subset size m= 7
Accuracy 0.814814814815
Random Forese with feature subset size m= 8
Accuracy 0.833833833834
```

```
In [283]: pred = rf[7].predict(Z)[0]
          with open('submission.txt','w') as f:
              for i in range(len(pred)):
                  f.write(str(int(pred[i])))
                  f.write('\n')
```

```
In [285]: common_split = []
          for i in range(rf[7].n):
              common_split.append([rf[7].feature[i][rf[7].decision_trees[i].tree_.feature[0]],rf
              #print(bt.decision_trees[i].tree_.threshold)
              #print(bt.decision_trees[i].tree_.feature)
          U,C = np.unique(common_split,axis=0,return_counts=True)
          print('Count\t\tSplits')
          print('------\t------------------------------')
          for i in range(len(U)):
              print(C[i],'\t',zfeatures[int(U[i,0])]+'\t<(>=)\t',str(U[i,1]))
```

```
Count                    Splits
------        ------------------------------
5           pclass        <(>=)         0.5
13           pclass        <(>=)         1.5
114           sex        <(>=)         0.5
1           age        <(>=)         1.5
1           age        <(>=)         5.5
1           age        <(>=)         15.5
2           sibsp        <(>=)         0.5
3           sibsp        <(>=)         1.5
3           parch        <(>=)         0.5
2           ticket        <(>=)         10.9207992554
1           ticket        <(>=)         10.9750003815
1           ticket        <(>=)         12.2354001999
2           ticket        <(>=)         12.28125
1           ticket        <(>=)         14.4791498184
1           ticket        <(>=)         15.1478996277
5           ticket        <(>=)         15.1729001999
4           ticket        <(>=)         15.6604499817
1           ticket        <(>=)         21.5499992371
1           ticket        <(>=)         51.9312515259
1           ticket        <(>=)         75.2458496094
1           ticket        <(>=)         77.6228942871
29           fare        <(>=)         0.5
7           cabin        <(>=)         1.5
```

```
In [295]: dataset = "spam"
          params = {
              "max_depth": 5,
              # "random_state": 6,
              "min_samples_leaf": 10,
          }
          N = 100
```

```
features = ["pain", "private", "bank", "money", "drug", "spam",
            "prescription", "creative", "height", "featured", "differ",
            "width", "other", "energy", "business", "message",
            "volumes", "revision", "path", "meter", "memo", "planning",
            "pleased", "record", "out", "semicolon", "dollar", "sharp",
            "exclamation", "parenthesis", "square_bracket", "ampersand"]
assert len(features) == 32

# Load spam data
path_train = 'spam_data.mat'
#path_test = 'datasets/spam_data/spam_test_labels.txt'
data = scipy.io.loadmat(path_train)
X = data['training_data']
y = np.squeeze(data['training_labels'])
#Z = data['test_data']
class_names = ["Ham", "Spam"]


print("Features", features)
print("Train size", X.shape)

print("\n\nPart 0: constant classifier")
print("Accuracy", 1 - np.sum(y) / y.size)

# Basic decision tree
print("\n\nPart (f-g): Bagged Random Forest")
m = [5,10,15,20,25,30,32]
rf = [RandomForest(params,m=m[i]) for i in range(len(m))]
for i in range(len(m)):
    np.random.seed(1)
    rf[i].fit(X, y)
    pred = rf[i].predict(X)
    print("Random Forese with feature subset size m=",m[i])
    print("Accuracy", 1 - np.mean(pred!=y))
```

```
Features ['pain', 'private', 'bank', 'money', 'drug', 'spam', 'prescription', 'creative', 'heigh
Train size (5172, 32)


Part 0: constant classifier
Accuracy 0.709976798144


Part (f-g): Bagged Random Forest
Random Forese with feature subset size m= 5
Accuracy 0.71094354215
Random Forese with feature subset size m= 10
```

```
Accuracy 0.723897911833
Random Forese with feature subset size m= 15
Accuracy 0.739752513534
Random Forese with feature subset size m= 20
Accuracy 0.763727764888
Random Forese with feature subset size m= 25
Accuracy 0.772041763341
Random Forese with feature subset size m= 30
Accuracy 0.816511987626
Random Forese with feature subset size m= 32
Accuracy 0.791376643465
```

1.7 (h)

```
In [330]: class BoostedRandomForest(RandomForest):

              def fit(self, X, y):
                  self.w = np.ones(X.shape[0]) / X.shape[0]  # Weights on data
                  self.a = np.zeros(self.n)  # Weights on decision trees
                  # TODO implement function
                  n1,n2 = np.shape(X)
                  for i in range(self.n):
                      index = np.random.choice(n1,n1)
                      num = np.random.choice(self.m,1)+1
                      fea = np.random.choice(n2,num,replace=False)
                      self.decision_trees[i].fit(X[index][:,fea],y[index])
                      self.feature.append(fea)
                      pred = self.decision_trees[i].predict(X[index][:,fea])
                      e = np.mean(pred!=y[index])
                      self.a[i] = 1/2*np.log((1-e)/e)
                      for j in range(self.m):
                          sign = -1 if pred[j]==y[index[j]] else 1
                          self.w[index[j]] *= np.exp(sign*self.a[i])

                  return self

              def predict(self, X):
                  # TODO implement function
                  result = np.zeros((self.n,np.shape(X)[0]))
                  for i in range(self.n):
                      result[i,:] = self.decision_trees[i].predict(X[:,self.feature[i]])
                  score = np.sum(result * self.a[:,np.newaxis],axis=0)
                  return np.array(score>= np.sum(self.a)/2,dtype=np.int16)
```

1.8 (i)

```
In [363]: with open('zdata','rb') as f:
              zdata = np.load(f)
          X = zdata[0]
          Z = zdata[1]
          y = zdata[2]
          print("Features", features)
          print("Train size", X.shape)
          # Basic decision tree
          print("Part (h-i): Boosted Random Forest")
          m = [1,2,3,4,5,6,7,8]
          brf = [BoostedRandomForest(params,m=m[i]) for i in range(len(m))]
          for i in range(len(m)):
              np.random.seed(1)
              brf[i].fit(X, y)
              pred = brf[i].predict(X)
              print("Random Forese with feature subset size m=",m[i])
              print("Accuracy", 1 - np.mean(pred!=y))
              #print("Predictions", pred[:100])
```

```
Features ['pain', 'private', 'bank', 'money', 'drug', 'spam', 'prescription', 'creative', 'heigh
Train size (999, 8)
Part (h-i): Boosted Random Forest
Random Forese with feature subset size m= 1
Accuracy 0.731731731732
Random Forese with feature subset size m= 2
Accuracy 0.76976976977
Random Forese with feature subset size m= 3
Accuracy 0.783783783784
Random Forese with feature subset size m= 4
Accuracy 0.800800800801
Random Forese with feature subset size m= 5
Accuracy 0.813813813814
Random Forese with feature subset size m= 6
Accuracy 0.831831831832
Random Forese with feature subset size m= 7
Accuracy 0.840840840841
Random Forese with feature subset size m= 8
Accuracy 0.835835835836
```

```
In [365]: print(Z)
```

```
[[  0.            0.           24.             0.             0.
   69.30000305    1.            1.        ]
 [  0.            0.           44.             0.             1.
   57.97919846    1.            1.        ]
 [  2.            1.            1.             5.             2.
```

```
   46.90000153    0.            3.        ]
 [   2.           1.           29.           0.           0.            7.875
     0.           3.        ]
 [   1.           1.           30.           0.           0.           13.
     0.           3.        ]
 [   2.           0.            9.           1.           1.
    15.24580002   0.            1.        ]
 [   1.           1.           20.           0.           0.
    13.86250019   1.            1.        ]
 [   1.           1.           25.           0.           0.           13.
     0.           3.        ]
 [   1.           1.           62.           0.           0.            9.6875
     0.           2.        ]
 [   0.           1.           39.           0.           0.
    29.70000076   1.            1.        ]
 [   0.           0.           42.           0.           0.
   227.5249939    0.            1.        ]
 [   1.           0.           24.           0.           0.           13.
     1.           3.        ]
 [   2.           0.            0.           1.           0.
    14.45419979   0.            1.        ]
 [   2.           0.            0.           3.           1.
    25.4666996    0.            3.        ]
 [   0.           0.           33.           0.           0.           86.5
     1.           3.        ]
 [   1.           0.            0.           0.           0.           21.
     0.           3.        ]
 [   2.           0.           15.           0.           0.
     7.2249999    0.            1.        ]
 [   0.           1.           24.           0.           0.
    79.19999695   1.            1.        ]
 [   2.           0.           37.           0.           0.
     9.58749962   0.            3.        ]
 [   2.           1.           40.           0.           0.
     7.89580011   0.            3.        ]
 [   0.           1.           13.           2.           2.          262.375
     1.           1.        ]
 [   1.           1.           24.           0.           0.           13.5
     0.           3.        ]
 [   2.           0.           30.           0.           0.
     7.62919998   0.            2.        ]
 [   0.           0.           24.           0.           0.
    69.30000305   1.            1.        ]
 [   2.           1.           18.           0.           0.
     7.7750001    0.            3.        ]
 [   0.           1.            0.           0.           0.
    29.70000076   1.            1.        ]
 [   2.           0.           18.           0.           0.            6.75
```

```
    0.          2.        ]
 [  1.          0.         34.          0.          0.         13.
    0.          3.        ]
 [  2.          0.         19.          0.          0.
    7.87919998  0.          2.        ]
 [  2.          1.          0.          0.          0.          7.75
    0.          2.        ]
 [  0.          1.         31.          1.          0.         52.
    1.          3.        ]
 [  2.          1.         25.          0.          0.
    7.89580011  0.          3.        ]
 [  2.          1.         32.          0.          0.
    7.85419989  0.          3.        ]
 [  1.          1.         57.          0.          0.
   12.35000038  0.          2.        ]
 [  2.          1.         22.          0.          0.
    7.89580011  0.          3.        ]
 [  2.          1.         51.          0.          0.
    7.05420017  0.          3.        ]
 [  1.          1.         26.          0.          0.         13.
    1.          3.        ]
 [  2.          1.         24.          2.          0.
   24.14999962  0.          3.        ]
 [  2.          1.         24.          0.          0.
    8.66250038  0.          3.        ]
 [  1.          1.         19.          0.          0.         10.5
    0.          3.        ]
 [  0.          1.         49.          1.          0.
   56.92919922  1.          1.        ]
 [  0.          0.          0.          1.          0.         52.
    1.          3.        ]
 [  2.          1.          0.          0.          0.         14.5
    0.          3.        ]
 [  2.          1.          0.          0.          0.          7.75
    0.          2.        ]
 [  1.          0.         40.          0.          0.         13.
    0.          3.        ]
 [  2.          1.          0.          0.          0.
   56.49580002  0.          3.        ]
 [  2.          1.          0.          0.          0.
    7.22919989  0.          1.        ]
 [  2.          1.         24.          0.          0.          9.5
    0.          3.        ]
 [  0.          0.         63.          1.          0.
   77.95829773  1.          3.        ]
 [  0.          1.         30.          0.          0.         27.75
    1.          1.        ]
 [  0.          1.          0.          0.          0.         31.
```

```
   0.          3.        ]
 [  2.          1.         11.          0.          0.
   18.78750038  0.          1.        ]
 [  2.          1.         32.          0.          0.
    8.05000019  0.          3.        ]
 [  1.          1.         41.          0.          0.         13.
    0.          3.        ]
 [  2.          0.         63.          0.          0.
    9.58749962  0.          3.        ]
 [  0.          0.          0.          1.          0.
   82.17079926  0.          1.        ]
 [  2.          1.         28.          2.          0.
    7.92500019  0.          3.        ]
 [  2.          0.         20.          0.          0.
    8.66250038  0.          3.        ]
 [  1.          1.         19.          0.          0.         10.5
    0.          3.        ]
 [  2.          1.         19.          0.          0.
   10.17080021  0.          3.        ]
 [  0.          1.          0.          0.          0.
   26.54999924  0.          3.        ]
 [  1.          0.         44.          1.          0.         26.
    0.          3.        ]
 [  2.          0.         43.          1.          6.
   46.90000153  0.          3.        ]
 [  1.          1.         18.          0.          0.         13.
    0.          3.        ]
 [  2.          0.          0.          0.          0.
    7.77920008  0.          2.        ]
 [  0.          0.         18.          1.          0.         60.
    1.          3.        ]
 [  2.          1.          0.          0.          0.
    7.22919989  0.          1.        ]
 [  2.          0.          0.          0.          0.          7.75
    0.          2.        ]
 [  2.          0.          0.          0.          0.
    7.73330021  0.          2.        ]
 [  2.          1.         24.          0.          0.
    7.49580002  0.          3.        ]
 [  2.          0.          0.          1.          0.         15.5
    0.          2.        ]
 [  1.          1.         32.          0.          0.         13.5
    0.          3.        ]
 [  2.          0.          0.          0.          0.
    7.62919998  0.          2.        ]
 [  2.          1.         19.          0.          0.
    8.1583004   0.          3.        ]
 [  2.          1.          0.          0.          0.
```

```
    7.22919989    0.            1.         ]
 [  0.            1.           46.            0.            0.
   75.24169922    1.            1.         ]
 [  2.            1.            0.            0.            0.            7.75
    0.            2.         ]
 [  2.            1.           32.5           0.            0.            9.5
    0.            3.         ]
 [  2.            1.           22.            0.            0.
    7.52080011    0.            3.         ]
 [  0.            0.           29.            0.            0.
  211.3374939     1.            3.         ]
 [  2.            1.           29.            3.            1.
   22.02499962    0.            3.         ]
 [  2.            1.           43.            0.            0.
    8.05000019    0.            3.         ]
 [  2.            0.            0.            1.            0.
   24.14999962    0.            2.         ]
 [  2.            1.           25.            0.            0.            7.25
    0.            3.         ]
 [  2.            1.           32.            0.            0.
   56.49580002    0.            3.         ]
 [  2.            1.            4.            1.            1.
   11.13329983    0.            3.         ]
 [  2.            1.           26.            0.            0.
    7.89580011    0.            3.         ]
 [  1.            1.           36.            0.            0.           13.
    0.            3.         ]
 [  1.            0.           17.            0.            0.           12.
    0.            1.         ]
 [  1.            1.           50.            1.            0.           26.
    0.            3.         ]
 [  1.            0.           33.            0.            2.           26.
    0.            3.         ]
 [  2.            1.           28.            0.            0.
   22.52499962    0.            3.         ]
 [  0.            1.           46.            0.            0.
   79.19999695    0.            1.         ]
 [  2.            0.            0.            2.            0.           23.25
    0.            2.         ]
 [  1.            1.           23.            2.            1.           11.5
    0.            3.         ]
 [  2.            1.            0.            0.            0.
    6.94999981    0.            2.         ]
 [  0.            0.           49.            0.            0.
   25.92919922    1.            3.         ]
 [  2.            1.            0.            1.            0.            7.75
    0.            2.         ]
 [  2.            1.           10.            4.            1.           29.125
```

```
    0.           2.         ]
 [  1.           1.          39.           0.           0.          13.
    0.           3.         ]
 [  2.           0.           9.           2.           2.          34.375
    0.           3.         ]
 [  2.           1.          29.           1.           0.
    7.04580021   0.           3.        ]
 [  1.           1.          40.           0.           0.          13.
    0.           3.         ]
 [  2.           1.           0.           0.           0.
    7.22919989   0.           1.        ]
 [  0.           0.          59.           2.           0.
   51.47919846   1.           3.        ]
 [  0.           0.          48.           1.           0.
  106.42500305   1.           1.        ]
 [  1.           0.          13.           0.           1.          19.5
    0.           3.         ]
 [  1.           0.          31.           0.           0.          21.
    0.           3.         ]
 [  1.           1.           1.           2.           1.          39.
    1.           3.         ]
 [  2.           0.          21.           1.           0.
    9.82499981   0.           3.        ]
 [  0.           1.          47.           0.           0.
   42.40000153   0.           3.        ]
 [  1.           1.          29.           0.           0.
   13.85830021   0.           1.        ]
 [  2.           0.          18.           0.           0.
    8.05000019   0.           3.        ]
 [  2.           1.           0.           0.           0.
    7.7750001    0.           3.        ]
 [  0.           0.          24.           0.           0.
   83.15830231   1.           1.        ]
 [  1.           0.          45.           0.           0.          13.5
    0.           3.         ]
 [  0.           0.          39.           1.           1.
   83.15830231   1.           1.        ]
 [  0.           1.          37.           1.           1.
   83.15830231   1.           1.        ]
 [  0.           1.          40.           0.           0.           0.
    1.           3.         ]
 [  0.           0.           0.           0.           0.
   79.19999695   0.           1.        ]
 [  1.           0.          12.           0.           0.          15.75
    0.           3.         ]
 [  1.           0.          50.           0.           0.          10.5
    0.           3.         ]
 [  1.           1.          43.           1.           1.          26.25
```

```
    0.            3.        ]
[   2.            0.           19.            1.            1.
   15.74170017    0.            1.        ]
[   1.            0.           29.            1.            0.           26.
    0.            3.        ]
[   2.            0.            0.            1.            2.
   23.45000076    0.            3.        ]
[   2.            1.           27.            0.            0.
    6.9749999     0.            3.        ]
[   1.            1.           32.            0.            0.           13.
    0.            3.        ]
[   2.            1.            0.            0.            0.
    7.22919989    0.            1.        ]
[   2.            1.           37.            2.            0.
    7.92500019    0.            3.        ]
[   1.            0.           36.            0.            3.           39.
    1.            3.        ]
[   2.            1.            0.            0.            0.
   14.45829964    0.            1.        ]
[   2.            1.            0.            0.            0.
    7.05000019    0.            3.        ]
[   0.            1.           45.5           0.            0.           28.5
    1.            3.        ]
[   2.            1.           25.            0.            0.
    7.6500001     1.            3.        ]
[   1.            1.           41.            0.            0.
   15.04580021    0.            1.        ]
[   2.            1.           22.            0.            0.
    8.05000019    0.            3.        ]
[   1.            0.            7.            0.            2.           26.25
    0.            3.        ]
[   0.            1.           47.            1.            0.
  227.5249939     1.            1.        ]
[   2.            1.           41.            0.            0.            7.125
    0.            3.        ]
[   0.            0.            0.            0.            0.
   31.68330002    0.            3.        ]
[   2.            0.            0.            0.            0.
    8.13749981    0.            2.        ]
[   0.            0.           21.            2.            2.          262.375
    1.            1.        ]
[   2.            1.           26.            1.            0.
    7.85419989    0.            3.        ]
[   2.            1.           22.5           0.            0.
    7.2249999     0.            1.        ]
[   2.            0.           23.            0.            0.
    7.92500019    0.            3.        ]
[   1.            1.           34.            0.            0.           13.
```

```
   0.           3.        ]
 [  2.           1.           0.           0.           0.
    7.89580011   0.           3.        ]
 [  1.           1.          54.           1.           0.          26.
    0.           3.        ]
 [  2.           0.          28.           0.           0.
    7.89580011   0.           3.        ]
 [  0.           0.          48.           1.           3.         262.375
    1.           1.        ]
 [  1.           1.          25.           0.           0.          13.
    0.           3.        ]
 [  2.           1.          30.           0.           0.
    8.05000019   0.           3.        ]
 [  2.           1.           9.           4.           2.
   31.38750076   0.           3.        ]
 [  0.           0.           0.           1.           0.
   89.10420227   1.           1.        ]
 [  2.           1.          25.           1.           0.
   17.79999924   0.           3.        ]
 [  1.           1.          29.           0.           0.          10.5
    0.           3.        ]
 [  0.           0.          50.           0.           0.
   28.71249962   1.           1.        ]
 [  2.           0.          21.           2.           2.          34.375
    0.           3.        ]
 [  2.           0.           2.           0.           1.
   10.46249962   1.           3.        ]
 [  2.           0.           0.           0.           0.
    8.05000019   0.           3.        ]
 [  2.           1.           0.           0.           0.
    7.2249999    0.           1.        ]
 [  2.           1.          19.           0.           0.          14.5
    0.           3.        ]
 [  1.           1.           8.           0.           2.          32.5
    0.           3.        ]
 [  0.           1.          48.           0.           0.
   26.54999924   1.           3.        ]
 [  1.           1.          34.           0.           0.          13.
    0.           3.        ]
 [  2.           1.          24.           0.           0.
    7.79580021   0.           3.        ]
 [  2.           1.           0.           0.           0.
    7.89580011   0.           3.        ]
 [  0.           1.           0.           0.           0.
   42.40000153   0.           3.        ]
 [  2.           1.           3.           4.           2.
   31.38750076   0.           3.        ]
 [  2.           1.          44.           0.           0.
```

```
    8.05000019    0.            3.        ]
[   1.            1.           28.            0.            0.            10.5
    0.            3.        ]
[   2.            0.           14.            0.            0.
    7.85419989    0.            3.        ]
[   0.            1.           58.            0.            0.
   29.70000076    1.            1.        ]
[   2.            1.            7.            1.            1.
   15.24580002    0.            1.        ]
[   2.            1.           24.            0.            0.
    7.89580011    0.            3.        ]
[   2.            1.            0.            0.            0.
    7.89580011    0.            3.        ]
[   2.            0.            0.            1.            0.            15.5
    0.            2.        ]
[   1.            0.           45.            1.            1.            26.25
    0.            3.        ]
[   2.            1.            6.            3.            1.
   21.07500076    0.            3.        ]
[   2.            0.            8.            3.            1.
   21.07500076    0.            3.        ]
[   2.            0.            5.            0.            0.
   12.47500038    0.            3.        ]
[   1.            0.           27.            1.            0.            21.
    0.            3.        ]
[   2.            1.            0.            0.            0.            14.5
    0.            3.        ]
[   2.            1.           16.            0.            0.
    9.2166996     0.            3.        ]
[   1.            0.           50.            0.            0.            10.5
    0.            3.        ]
[   2.            1.           17.            0.            0.
    8.66250038    0.            3.        ]
[   1.            1.           27.            0.            0.
   15.0333004     0.            1.        ]
[   0.            0.           22.            0.            2.            49.5
    1.            1.        ]
[   1.            1.           19.            0.            0.            10.5
    0.            3.        ]
[   2.            0.            9.            3.            2.
   27.89999962    0.            3.        ]
[   2.            1.           36.            0.            0.             7.25
    0.            3.        ]
[   2.            1.           38.            0.            0.
    7.05000019    0.            3.        ]
[   2.            0.            9.            4.            2.
   31.27499962    0.            3.        ]
[   0.            1.           35.            0.            0.
```

```
   26.28750038    1.            3.        ]
[   2.            1.           21.            0.            0.            7.25
    0.            3.         ]
[   1.            0.           30.            3.            0.           21.
    0.            3.         ]
[   0.            1.           44.            2.            0.           90.
    1.            2.         ]
[   2.            0.           22.            0.            0.            7.75
    0.            3.         ]
[   2.            1.           32.            0.            0.
    7.92500019    0.            3.        ]
[   0.            1.            0.            0.            0.
   27.7208004     0.            1.        ]
[   2.            0.           13.            0.            0.
    7.22919989    0.            1.        ]
[   2.            1.            0.            0.            0.            7.75
    0.            2.         ]
[   0.            1.           23.            0.            1.
   63.35829926    1.            1.        ]
[   2.            1.           28.5           0.            0.
   16.10000038    0.            3.        ]
[   1.            0.           18.            1.            1.           13.
    0.            3.         ]
[   2.            0.            2.            4.            2.
   31.27499962    0.            3.        ]
[   0.            0.            0.            0.            0.
   27.7208004     0.            1.        ]
[   0.            0.           55.            2.            0.
   25.70000076    1.            3.        ]
[   1.            1.           50.            0.            0.           13.
    0.            3.         ]
[   0.            1.           45.            0.            0.
   29.70000076    1.            1.        ]
[   1.            1.           27.            1.            0.           26.
    0.            3.         ]
[   2.            0.           28.            0.            0.
    7.7750001     0.            3.        ]
[   2.            1.           22.            1.            0.            7.25
    0.            3.         ]
[   2.            0.            0.            3.            1.
   25.4666996     0.            3.        ]
[   2.            1.            1.            1.            2.
   20.57500076    0.            3.        ]
[   1.            1.           25.            0.            0.           10.5
    0.            3.         ]
[   2.            1.            0.            0.            0.
    7.87919998    0.            2.        ]
[   0.            1.           39.            1.            0.
```

```
   71.28330231    1.            1.        ]
 [   1.           1.           28.          0.           0.           13.5
     0.           3.        ]
 [   0.           1.           25.          1.           0.
    55.44169998   1.            1.        ]
 [   2.           1.           40.5         0.           0.
    15.10000038   0.            3.        ]
 [   1.           1.           46.          0.           0.           26.
     0.           3.        ]
 [   1.           0.           29.          1.           0.           26.
     0.           3.        ]
 [   2.           1.           32.          0.           0.
    22.52499962   0.            3.        ]
 [   0.           0.           58.          0.           0.
   146.52079773   1.            1.        ]
 [   0.           1.            0.          0.           0.           52.
     1.           3.        ]
 [   2.           1.           12.          1.           0.
    11.24170017   0.            1.        ]
 [   2.           1.           25.          0.           0.
     7.05000019   0.            3.        ]
 [   2.           1.           18.          0.           0.
     8.66250038   0.            3.        ]
 [   2.           1.           18.          0.           0.
     8.30000019   0.            3.        ]
 [   2.           0.            4.          0.           1.
    13.41670036   0.            1.        ]
 [   2.           0.           30.          0.           0.
    12.47500038   0.            3.        ]
 [   2.           0.            0.          0.           0.           14.5
     0.           3.        ]
 [   2.           1.            0.33329999  0.           2.
    14.39999962   0.            3.        ]
 [   0.           0.           29.          0.           0.
   221.77920532   1.            3.        ]
 [   0.           0.           36.          0.           0.
    31.67919922   1.            1.        ]
 [   1.           1.           23.          0.           0.           13.
     0.           3.        ]
 [   1.           0.           57.          0.           0.           10.5
     1.           3.        ]
 [   2.           0.           38.          0.           0.
     7.22919989   0.            1.        ]
 [   2.           0.           30.          0.           0.
     8.66250038   0.            3.        ]
 [   2.           1.            9.          0.           2.
    20.52499962   0.            3.        ]
 [   0.           0.            0.          0.           1.           55.
```

```
    1.           3.          ]
[   0.           0.          22.          0.          1.
   59.40000153   0.           1.          ]
[   1.           1.           0.          0.          0.           0.
    0.           3.          ]
[   2.           0.           0.          6.          2.
   69.55000305   0.           3.          ]
[   2.           1.          28.          0.          0.
    7.89580011   0.           3.          ]
[   0.           0.          16.          0.          0.          86.5
    1.           3.          ]
[   2.           1.           0.          1.          0.          15.5
    0.           2.          ]
[   2.           0.           0.          0.          2.
   15.24580002   0.           1.          ]
[   1.           1.          23.          0.          0.          10.5
    0.           3.          ]
[   0.           0.          18.          1.          0.
  227.5249939    1.           1.          ]
[   2.           1.          24.          1.          0.
   16.10000038   0.           3.          ]
[   2.           0.          10.          0.          2.
   24.14999962   0.           3.          ]
[   2.           1.          28.          0.          0.
   22.52499962   0.           3.          ]
[   0.           1.          47.          0.          0.
   25.58749962   1.           3.          ]
[   2.           1.          28.          1.          0.
   15.85000038   0.           3.          ]
[   2.           1.          18.          1.          1.
   20.21249962   0.           3.          ]
[   1.           1.          21.          2.          0.          73.5
    0.           3.          ]
[   2.           0.           0.75        2.          1.
   19.25830078   0.           1.          ]
[   2.           1.           0.          1.          0.
   14.45419979   0.           1.          ]
[   2.           1.          40.5         0.          0.           7.75
    0.           2.          ]
[   2.           1.          22.          0.          0.
    7.22919989   0.           1.          ]
[   2.           0.           0.          0.          0.
    7.72079992   0.           2.          ]
[   0.           0.          30.          0.          0.
   56.92919922   1.           1.          ]
[   2.           1.           0.          0.          0.           7.75
    0.           2.          ]
[   2.           1.          39.          1.          5.
```

```
  31.27499962    0.           3.        ]
 [  1.           1.          14.           0.           0.          65.
    0.           3.        ]
 [  2.           1.          32.           0.           0.
    8.05000019   1.           3.        ]
 [  1.           0.          24.           1.           1.
   37.00419998   0.           1.        ]
 [  1.           0.          50.           0.           1.          26.
    0.           3.        ]
 [  2.           1.          21.           0.           0.
    8.43330002   0.           3.        ]
 [  0.           0.          16.           0.           1.
   57.97919846   1.           1.        ]
 [  2.           1.          34.5          0.           0.
    7.82919979   0.           2.        ]
 [  2.           0.          45.           1.           0.
   14.10830021   0.           3.        ]
 [  0.           1.           0.           0.           0.          26.
    0.           3.        ]
 [  2.           1.          20.           0.           0.
    7.22919989   0.           1.        ]
 [  2.           1.          26.           0.           0.
    7.7750001    0.           3.        ]
 [  2.           0.          27.           0.           0.
    7.92500019   0.           3.        ]
 [  2.           1.          41.           0.           0.
    7.8499999    0.           3.        ]
 [  2.           0.           0.           0.           2.           7.75
    0.           2.        ]
 [  2.           1.          18.           1.           0.
   14.45419979   0.           1.        ]
 [  2.           1.          28.5          0.           0.
    7.22919989   0.           1.        ]
 [  2.           1.           0.           1.           0.
   16.10000038   0.           3.        ]
 [  2.           1.          19.           0.           0.
    7.89580011   0.           3.        ]
 [  0.           1.          28.           0.           0.          35.5
    1.           3.        ]
 [  2.           1.           0.           0.           0.           7.25
    0.           3.        ]
 [  0.           1.          50.           1.           0.
  106.42500305   1.           1.        ]
 [  0.           1.           0.           0.           0.          35.
    1.           3.        ]
 [  0.           1.          33.           0.           0.
   26.54999924   0.           3.        ]
 [  2.           1.          39.           0.           1.
```

```
   13.41670036    0.           1.         ]
 [   2.           1.          27.           0.            0.
     8.66250038   0.           3.         ]
 [   1.           1.          44.           0.            0.           13.
     0.           3.         ]
 [   0.           1.          49.           1.            0.
    89.10420227   1.           1.         ]
 [   2.           1.           0.           2.            0.
    21.67919922   0.           1.         ]
 [   1.           0.          27.           0.            0.           10.5
     1.           3.         ]
 [   1.           0.          12.           2.            1.           39.
     1.           3.         ]
 [   2.           1.          23.           0.            0.
     7.79580021   0.           3.         ]
 [   2.           0.          24.           1.            0.
    15.85000038   0.           3.         ]
 [   0.           1.          26.           0.            0.           30.
     1.           1.         ]
 [   2.           0.           6.           4.            2.
    31.27499962   0.           3.         ]
 [   2.           0.           0.           0.            0.            7.75
     0.           2.         ]
 [   0.           0.          50.           1.            1.          211.5
     1.           1.         ]
 [   2.           1.          21.           0.            0.
     8.05000019   0.           3.         ]
 [   2.           1.           0.           0.            0.
     7.57499981   0.           3.         ]
 [   1.           1.           0.           0.            0.
    15.04580021   1.           1.         ]
 [   1.           0.          30.           0.            0.           13.
     0.           3.         ]
 [   0.           1.          42.           0.            0.
    26.28750038   1.           3.         ]
 [   2.           1.           0.           0.            0.            7.75
     0.           2.         ]
 [   1.           1.          35.           0.            0.
    12.35000038   0.           2.         ]]
```

```
In [364]: pred = brf[6].predict(Z)
          with open('submission.txt','w') as f:
              for i in range(len(pred)):
                  f.write(str(int(pred[i])))
                  f.write('\n')
```

```
In [255]: brf[3].a
```

```
Out[255]: array([ 0.45919689,  0.50171324,  0.70195244,  0.46165415,  0.75418265,
                  0.67703674,  0.71149206,  0.55936685,  0.81743921,  0.50426593,
                  0.77816404,  0.79933291,  0.46658438,  0.75418265,  0.50171324,
                  0.5093896 ,  0.4567448 ,  0.68940184,  0.79576349,  0.73419586,
                  0.74080227,  0.76779533,  0.69565345,  0.46658438,  0.55398845,
                  0.78515511,  0.49156221,  0.51971171,  0.7712361 ,  0.54066806,
                  0.51453804,  0.81014396,  0.73091305,  0.50426593,  0.73419586,
                  0.73091305,  0.74412618,  0.67397353,  0.52752052,  0.70512003,
                  0.77469228,  0.46411663,  0.7508162 ,  0.76095927,  0.81014396,
                  0.46411663,  0.68940184,  0.75418265,  0.71791415,  0.76779533,
                  0.70195244,  0.58949114,  0.51453804,  0.46411663,  0.72438742,
                  0.50171324,  0.50171324,  0.48903911,  0.5093896 ,  0.50171324,
                  0.5119607 ,  0.6709213 ,  0.55936685,  0.53538874,  0.43730902,
                  0.49409111,  0.52491106,  0.49409111,  0.53013658,  0.8397671 ,
                  0.79576349,  0.48903911,  0.58394321,  0.73749222,  0.5068247 ,
                  0.41344207,  0.76095927,  0.71791415,  0.77469228,  0.57842744,
                  0.7712361 ,  0.74080227,  0.72114432,  0.52230814,  0.63511109,
                  0.6709213 ,  0.77816404,  0.7712361 ,  0.47401972,  0.84736369,
                  0.47900382,  0.5068247 ,  0.69565345,  0.81743921,  0.49409111,
                  0.73749222,  0.82480687,  0.70829986,  0.54863903,  0.80291931,
                  0.81743921,  0.67397353,  0.5068247 ,  0.70512003,  0.73419586,
                  0.5093896 ,  0.68629349,  0.51971171,  0.79221086,  0.73091305,
                  0.44941914,  0.55936685,  0.71149206,  0.68629349,  0.5093896 ,
                  0.76779533,  0.73091305,  0.54331805,  0.73091305,  0.56748997,
                  0.49916658,  0.7508162 ,  0.51712169,  0.74080227,  0.79221086,
                  0.7508162 ,  0.46165415,  0.47153588,  0.72114432,  0.6831966 ,
                  0.72114432,  0.45919689,  0.49156221,  0.72114432,  0.52230814,
                  0.80652293,  0.4349015 ,  0.5093896 ,  0.83224884,  0.43972131,
                  0.87064502,  0.56206708,  0.39933449,  0.77469228,  0.48150418,
                  0.82111387,  0.52752052,  0.80652293,  0.85504072,  0.47153588,
                  0.42293413,  0.79933291,  0.7508162 ,  0.45185597,  0.52752052,
                  0.69252179,  0.48652177,  0.7474641 ,  0.4277072 ,  0.61773574,
                  0.65881998,  0.45919689,  0.43249872,  0.7712361 ,  0.7508162 ,
                  0.71149206,  0.6831966 ,  0.48401015,  0.86671195,  0.45429785,
                  0.7508162 ,  0.49409111,  0.46658438,  0.48903911,  0.68629349,
                  0.76095927,  0.80652293,  0.76095927,  0.46905745,  0.50171324,
                  0.72114432,  0.49916658,  0.68629349,  0.6618294 ,  0.7474641 ,
                  0.77469228,  0.72764362,  0.7712361 ,  0.85891009,  0.58394321,
                  0.69252179,  0.48652177,  0.74412618,  0.46165415,  0.63219175,
                  0.46658438,  0.46411663,  0.78515511,  0.4567448 ,  0.70829986])
```

```
In [361]: dataset = "spam"
          params = {
              "max_depth": 5,
              # "random_state": 6,
              "min_samples_leaf": 10,
          }
          N = 100
```

```python
features = ["pain", "private", "bank", "money", "drug", "spam",
            "prescription", "creative", "height", "featured", "differ",
            "width", "other", "energy", "business", "message",
            "volumes", "revision", "path", "meter", "memo", "planning",
            "pleased", "record", "out", "semicolon", "dollar", "sharp",
            "exclamation", "parenthesis", "square_bracket", "ampersand"]
assert len(features) == 32

# Load spam data
path_train = 'spam_data.mat'
#path_test = 'datasets/spam_data/spam_test_labels.txt'
data = scipy.io.loadmat(path_train)
X = data['training_data']
y = np.squeeze(data['training_labels'])
Z = data['test_data']
class_names = ["Ham", "Spam"]


print("Features", features)
print("Train size", X.shape)

print("\n\nPart 0: constant classifier")
print("Accuracy", 1 - np.sum(y) / y.size)

# Basic decision tree
print("\n\nPart (h-i): Boosted Random Forest")
m = [5,10,15,20,15,30,32]
brf = [BoostedRandomForest(params,m=m[i]) for i in range(len(m))]
for i in range(len(m)):
    np.random.seed(1)
    brf[i].fit(X, y)
    pred = brf[i].predict(X)
    print("Random Forese with feature subset size m=",m[i])
    print("Accuracy", 1 - np.mean(pred!=y))
#print("Predictions", pred[:100])
```

```
Features ['pain', 'private', 'bank', 'money', 'drug', 'spam', 'prescription', 'creative', 'heigh
Train size (5172, 32)


Part 0: constant classifier
Accuracy 0.709976798144


Part (h-i): Boosted Random Forest
Random Forese with feature subset size m= 5
Accuracy 0.712103634957
```

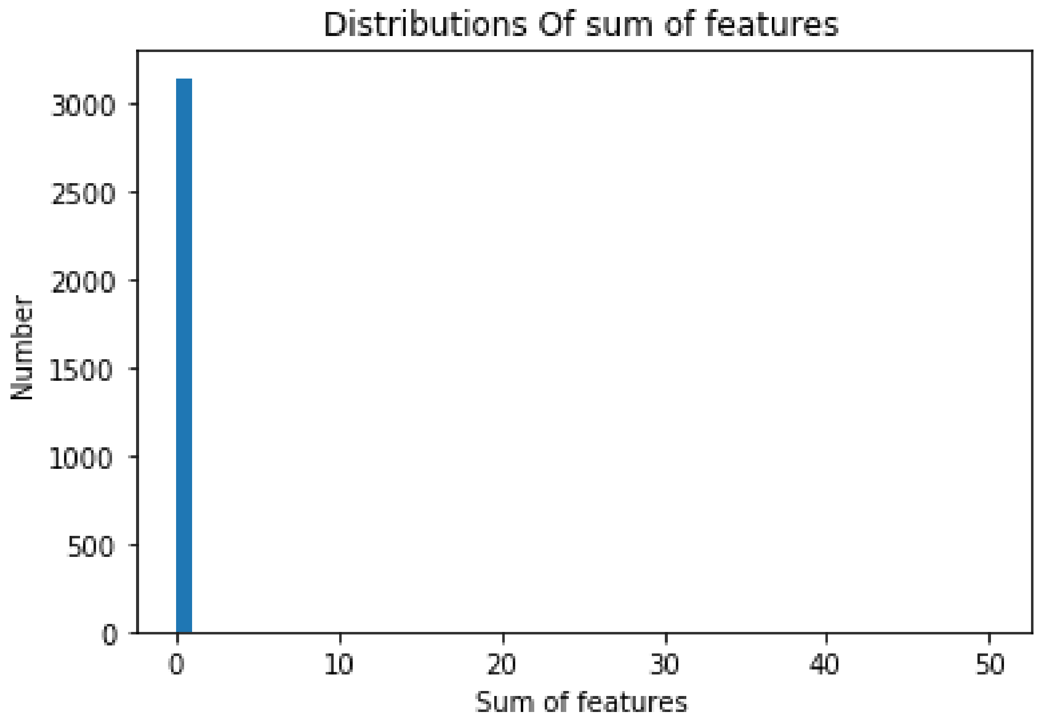
```
Random Forese with feature subset size m= 10
Accuracy 0.727184841454
Random Forese with feature subset size m= 15
Accuracy 0.747873163186
Random Forese with feature subset size m= 20
Accuracy 0.775522041763
Random Forese with feature subset size m= 15
Accuracy 0.747873163186
Random Forese with feature subset size m= 30
Accuracy 0.815351894818
Random Forese with feature subset size m= 32
Accuracy 0.816705336427
```

```
In [362]: pred = brf[6].predict(Z)
          with open('submission.txt','w') as f:
              for i in range(len(pred)):
                  f.write(str(int(pred[i])))
                  f.write('\n')
```

```
In [360]: np.set_printoptions(threshold=np.inf)
          a1 = np.sum(X[pred!=y,:],axis=1)
          a1.dtype = np.int16
          X[pred!=y,:][:3,:]
```

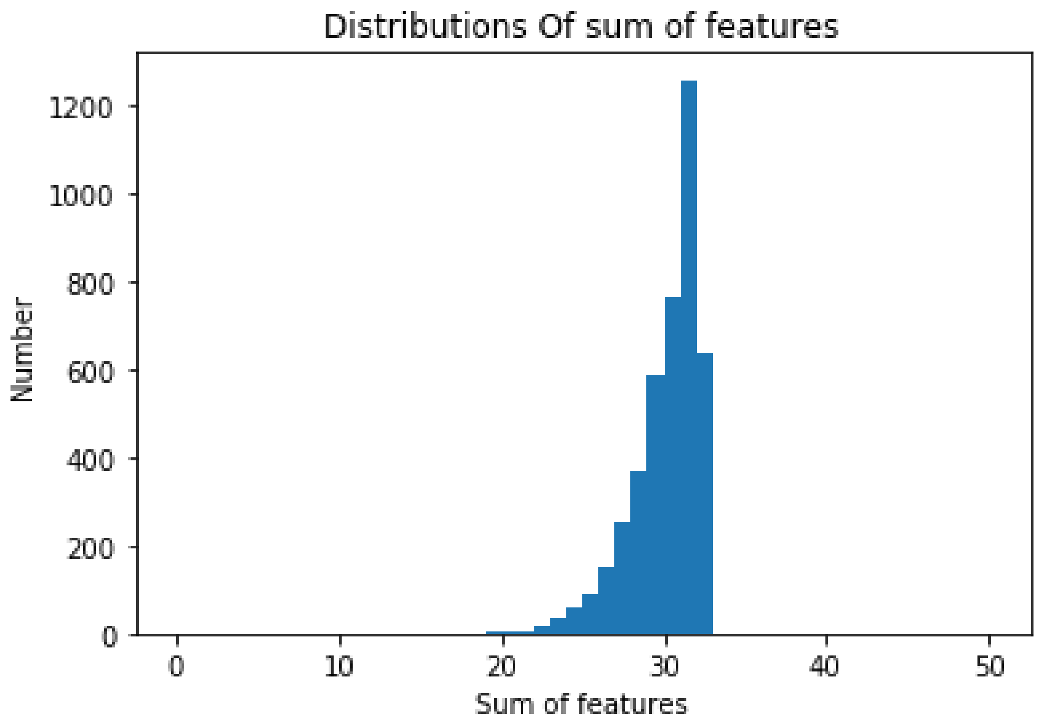
```
Out[360]: array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                   0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                   0.,  0.,  0.,  0.,  0.,  0.],
                 [ 0.,  0.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                   0.,  3.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  3.,
                   1.,  0.,  0.,  2.,  0.,  0.],
                 [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,
                   0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                   1.,  0.,  0.,  2.,  0.,  0.]])
```

```
In [367]: import matplotlib.pyplot as plt
          plt.hist(a1, 50,range=(0,50))
          plt.xlabel('Sum of features')
          plt.ylabel('Number')
          plt.title('Distributions Of sum of features')
          plt.show()
```



```
In [355]: a2 = np.sum(X[pred==y,:]==0,axis=1)
```

```
In [368]: plt.hist(a2, 50, range=(0,50))
          plt.xlabel('Sum of features')
          plt.ylabel('Number')
          plt.title('Distributions Of sum of features')
          plt.show()
```



1.9 (j)

```
In [397]: from sklearn.model_selection import KFold
          kf = KFold(n_splits=3,shuffle=True)

In [398]: with open('zdata','rb') as f:
              zdata = np.load(f)
          X = zdata[0]
          y = zdata[2]
          i = 1
          params = {
              "max_depth": 5,
              # "random_state": 6,
              "min_samples_leaf": 10,
          }

          for train, test in kf.split(y):
              print('---------- Data Split '+str(i)+' -----------')
              i += 1
              print("***   A single decision tree \t***")
              dt = DecisionTree(max_depth=3, feature_labels=features)
              dt.fit(X[train], y[train])
              print("Training Accuracy:", 1 - np.mean(dt.predict(X[train])!=y[train]))
              print("Testing  Accuracy:", 1 - np.mean(dt.predict(X[test])!=y[test]))
```

```
        print("\n***       Bagged trees   \t***")
        bt = BaggedTrees(params)
        bt.fit(X[train], y[train])
        print("Training Accuracy:", 1 - np.mean(dt.predict(X[train])!=y[train]))
        print("Testing  Accuracy:", 1 - np.mean(dt.predict(X[test])!=y[test]))

        print("\n***      Random forests   \t***")
        rf = RandomForest(params,m=8)
        rf.fit(X[train], y[train])
        print("Training Accuracy:", 1 - np.mean(rf.predict(X[train])!=y[train]))
        print("Testing  Accuracy:", 1 - np.mean(rf.predict(X[test])!=y[test]))

        print("\n***   Boosted Random forests   \t***")
        brf = BoostedRandomForest(params,m=7)
        brf.fit(X[train], y[train])
        print("Training Accuracy:", 1 - np.mean(brf.predict(X[train])!=y[train]))
        print("Testing  Accuracy:", 1 - np.mean(brf.predict(X[test])!=y[test]))

        print('\n')
```

```
---------- Data Split 1 -----------
***   A single decision tree         ***
Training Accuracy: 0.807807807808
Testing  Accuracy: 0.774774774775

***        Bagged trees           ***
Training Accuracy: 0.807807807808
Testing  Accuracy: 0.774774774775

***       Random forests          ***
Training Accuracy: 0.833333333333
Testing  Accuracy: 0.792792792793

***    Boosted Random forests          ***
Training Accuracy: 0.836336336336
Testing  Accuracy: 0.783783783784



---------- Data Split 2 -----------
***   A single decision tree         ***
Training Accuracy: 0.782282282282
Testing  Accuracy: 0.84984984985

***        Bagged trees           ***
Training Accuracy: 0.782282282282
Testing  Accuracy: 0.84984984985
```

```
***       Random forests          ***
Training Accuracy: 0.816816816817
Testing  Accuracy: 0.831831831832

***    Boosted Random forests          ***
Training Accuracy: 0.816816816817
Testing  Accuracy: 0.855855855856


---------- Data Split 3 -----------
***   A single decision tree         ***
Training Accuracy: 0.824324324324
Testing  Accuracy: 0.774774774775

***        Bagged trees           ***
Training Accuracy: 0.824324324324
Testing  Accuracy: 0.774774774775

***       Random forests          ***
Training Accuracy: 0.872372372372
Testing  Accuracy: 0.753753753754

***    Boosted Random forests          ***
Training Accuracy: 0.869369369369
Testing  Accuracy: 0.777777777778
```

```
In [401]: dataset = "spam"
          params = {
              "max_depth": 5,
              # "random_state": 6,
              "min_samples_leaf": 10,
          }
          N = 100

          features = ["pain", "private", "bank", "money", "drug", "spam",
                      "prescription", "creative", "height", "featured", "differ",
                      "width", "other", "energy", "business", "message",
                      "volumes", "revision", "path", "meter", "memo", "planning",
                      "pleased", "record", "out", "semicolon", "dollar", "sharp",
                      "exclamation", "parenthesis", "square_bracket", "ampersand"]
          assert len(features) == 32

          # Load spam data
          path_train = 'spam_data.mat'
          #path_test = 'datasets/spam_data/spam_test_labels.txt'
```

```
        data = scipy.io.loadmat(path_train)
        X = data['training_data']
        y = np.squeeze(data['training_labels'])
        Z = data['test_data']
        class_names = ["Ham", "Spam"]
        i = 1
        for train, test in kf.split(y):
            print('---------- Data Split '+str(i)+' -----------')
            i += 1
            print("***   A single decision tree \t***")
            dt = DecisionTree(max_depth=3, feature_labels=features)
            dt.fit(X[train], y[train])
            print("Training Accuracy:", 1 - np.mean(dt.predict(X[train])!=y[train]))
            print("Testing  Accuracy:", 1 - np.mean(dt.predict(X[test])!=y[test]))

            print("\n***        Bagged trees   \t***")
            bt = BaggedTrees(params)
            bt.fit(X[train], y[train])
            print("Training Accuracy:", 1 - np.mean(dt.predict(X[train])!=y[train]))
            print("Testing  Accuracy:", 1 - np.mean(dt.predict(X[test])!=y[test]))

            print("\n***       Random forests  \t***")
            rf = RandomForest(params,m=30)
            rf.fit(X[train], y[train])
            print("Training Accuracy:", 1 - np.mean(rf.predict(X[train])!=y[train]))
            print("Testing  Accuracy:", 1 - np.mean(rf.predict(X[test])!=y[test]))

            print("\n***    Boosted Random forests  \t***")
            brf = BoostedRandomForest(params,m=32)
            brf.fit(X[train], y[train])
            print("Training Accuracy:", 1 - np.mean(brf.predict(X[train])!=y[train]))
            print("Testing  Accuracy:", 1 - np.mean(brf.predict(X[test])!=y[test]))

            print('\n')
```

```
---------- Data Split 1 -----------
***   A single decision tree         ***
Training Accuracy: 0.797853828306
Testing  Accuracy: 0.788863109049

***        Bagged trees           ***
Training Accuracy: 0.797853828306
Testing  Accuracy: 0.788863109049

***       Random forests          ***
Training Accuracy: 0.792343387471
Testing  Accuracy: 0.780742459397
```

```
***    Boosted Random forests          ***
Training Accuracy: 0.81873549884
Testing  Accuracy: 0.807424593968


---------- Data Split 2 -----------
***   A single decision tree        ***
Training Accuracy: 0.79379350348
Testing  Accuracy: 0.796983758701

***        Bagged trees           ***
Training Accuracy: 0.79379350348
Testing  Accuracy: 0.796983758701

***       Random forests          ***
Training Accuracy: 0.779582366589
Testing  Accuracy: 0.772041763341

***    Boosted Random forests          ***
Training Accuracy: 0.817575406032
Testing  Accuracy: 0.813225058005


---------- Data Split 3 -----------
***   A single decision tree        ***
Training Accuracy: 0.792923433875
Testing  Accuracy: 0.798723897912

***        Bagged trees           ***
Training Accuracy: 0.792923433875
Testing  Accuracy: 0.798723897912

***       Random forests          ***
Training Accuracy: 0.782192575406
Testing  Accuracy: 0.79060324826

***    Boosted Random forests          ***
Training Accuracy: 0.819315545244
Testing  Accuracy: 0.817865429234
```