1. (a)

I did this homework with Luning Zhao.

We work on our homework seperately, but we discuss when meeting problems.

Homework is fine.

(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.

Siyao Jia

2. (a) $J_\lambda(\vec{w}) = \frac{1}{2}(\vec{y} - X\vec{w})^T(\vec{y} - X\vec{w}) + \lambda\|\vec{w}\|$

$= \frac{1}{2}\|\vec{y}\|^2 - \vec{y}^T X\vec{w} + \vec{w}^T X^T X\vec{w} + \lambda\|\vec{w}\|$

$= \frac{1}{2}\|\vec{y}\|^2 - \sum_{i=1}^{d} \vec{y}^T\vec{x}_i w_i + \|\vec{w}\|^2 + \lambda\|\vec{w}\|$

$= \frac{1}{2}\|\vec{y}\|^2 + \sum_{i=1}^{d}(w_i^2 + \lambda|w_i| - \vec{y}^T\vec{x}_i w_i)$

$= g(\vec{y}) + \sum_{i=1}^{d} f(\vec{x}_i, \vec{y}, w_i, \lambda)$ where $g(\vec{y}) = \frac{1}{2}\|\vec{y}\|^2$

$f(\vec{x}_i, \vec{y}, w_i, \lambda) = w_i^2 + \lambda|w_i| - \vec{y}^T\vec{x}_i w_i$

So $\hat{w} = \arg\min_{\vec{w}} \{J_\lambda(\vec{w})\} = \arg\min_{\vec{w}} \sum_{i=1}^{d} f(\vec{x}_i, \vec{y}, w_i, \lambda)$

so the joint optimization can be decomposed into individual optimizations.

so we can learn w_i by $\arg\min_{w_i} f(\vec{x}_i, \vec{y}, w_i, \lambda)$

(b) If $\hat{w}_i > 0$, $f(\vec{x}_i, \vec{y}, w_i, \lambda) = w_i^2 + \lambda w_i - \vec{y}^T\vec{x}_i w_i$

f is minimized when $\hat{w}_i = -\frac{\lambda - \vec{y}^T\vec{x}_i}{2} = \frac{-\lambda + \vec{y}^T\vec{x}_i}{2}$ (when $\lambda < \vec{y}^T\vec{x}_i$)

(c) If $\hat{w}_i < 0$, $f(\vec{x}_i, \vec{y}, w_i, \lambda) = w_i^2 - \lambda w_i - \vec{y}^T\vec{x}_i w_i$

f is minimized when $\hat{w}_i = -\frac{-\lambda - \vec{y}^T\vec{x}_i}{2} = \frac{\lambda + \vec{y}^T\vec{x}_i}{2}$ (when $\lambda < -\vec{y}^T\vec{x}_i$)

(d) when $\lambda \in [-\vec{y}^T\vec{x}_i, \vec{y}^T\vec{x}_i]$, neither (b) or (c) works, so $\hat{w}_i = 0$

(e) Similarly, $f(\vec{x}_i, \vec{y}, w_i, \lambda) = w_i^2 + \lambda w_i^2 - \vec{y}^T\vec{x}_i w_i$

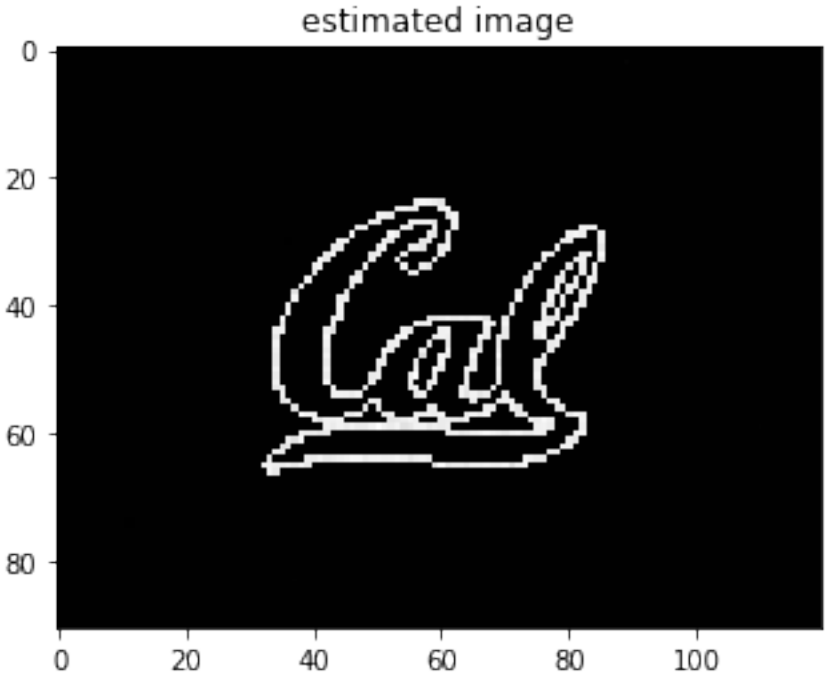
$= (1+\lambda) w_i^2 - \vec{y}^T\vec{x}_i w_i$

$f(\vec{x}_i, \vec{y}, w_i, \lambda)$ is minimized when $w_i = -\frac{\vec{y}^T\vec{x}_i}{2(1+\lambda)} = \frac{\vec{y}^T\vec{x}_i}{2(1+\lambda)}$

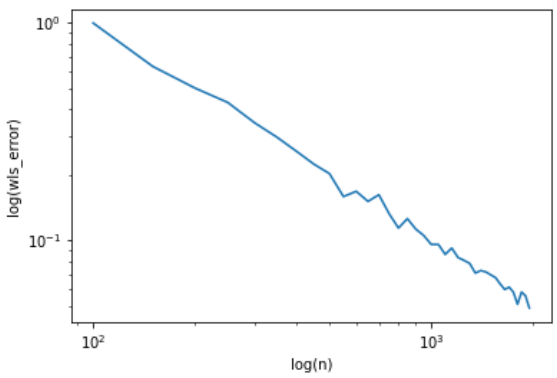
$\hat{w}_i = 0$ only when $\vec{y}^T\vec{x}_i = 0$, which is very hard to obtain compare with (d).

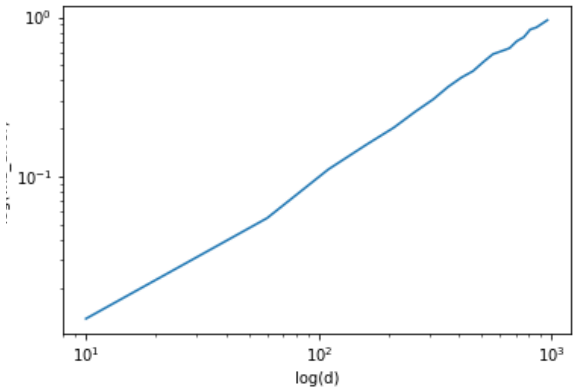
So l_1 norm promotes sparsity.

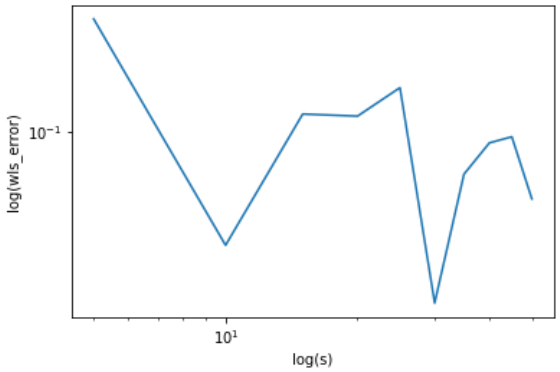
2 (f) let lambda = 0.00001, and run LASSO(imDims, measurements, X, 0.00001).



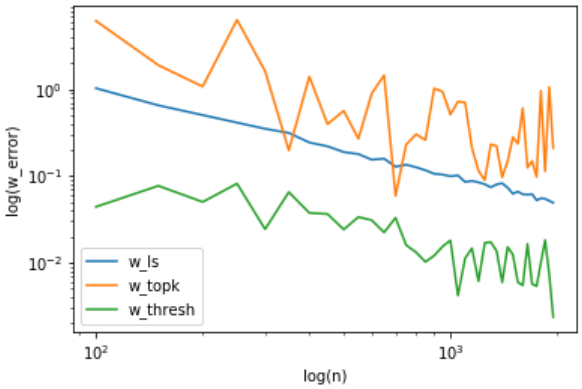
3 (a)

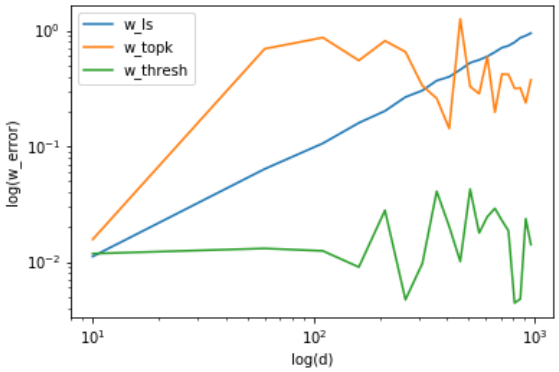

log(wls_error)
log(n)

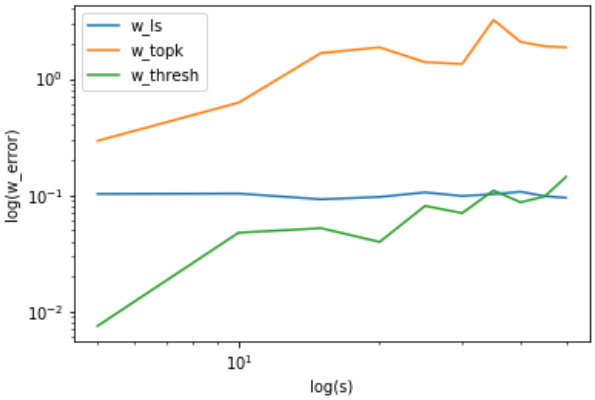

log(d)


log(wls_error)
log(s)

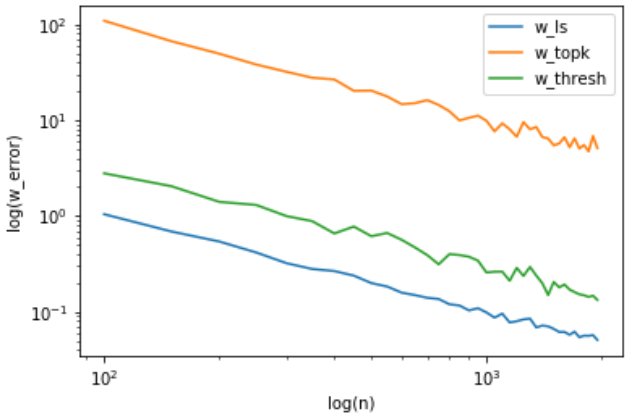
(b)

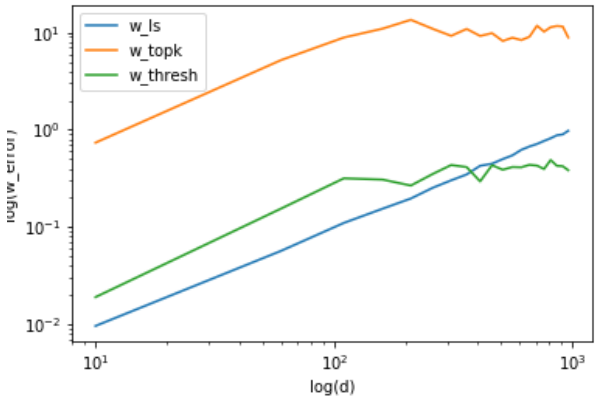

log(w_error)
w_ls
w_topk
w_thresh
log(n)

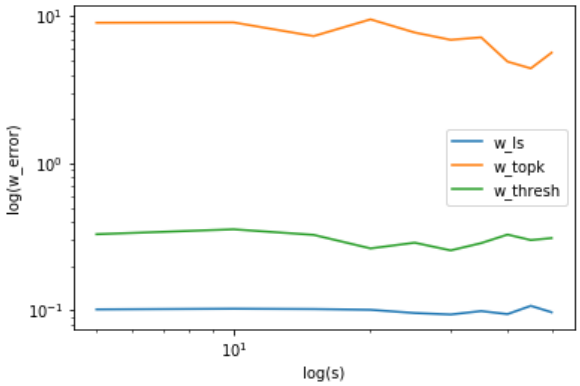

log(w_error)
w_ls
w_topk
w_thresh
log(d)


log(w_error)
w_ls
w_topk
w_thresh
log(s)

(c)







For sparse linear model, threshold/topk sparsity estimator has smaller variance, and its bias is not very large, so it has small total errors as shown in (b).
But for non-sparse linear model, threshold/topk sparsity estimator has very large bias, which results in large total errors as shown in (c).
Topk model has so large bias that it always has larger error than OLS, but threshold sparsity model works even better than OLS for sparse linear model.

3 (d) $\Pr\{\max_i |z_i| \geq 2\sigma\sqrt{\log d}\}$

$\geq \sum_{i=1}^{d} \Pr\{|z_i| \geq 2\sigma\sqrt{\log d}\}$

$\geq d\, e^{-\frac{4\sigma^2 \log d}{2\sigma^2}}$

$= d \cdot (e^{\log d})^{-2}$

$= \frac{1}{d}$

So we prove that $\Pr\{\max_i |z_i| \geq 2\sigma\sqrt{\log d}\} \leq \frac{1}{d}$.

(e) $\hat{w}_{LS} = X^T y = X^T(Xw^* + z) = X^T X w^* + X^T z$

Because X is orthonormal, $X^T X = I$

$\mathrm{Cov}(X^T z) = X\,\mathrm{Cov}(z)\,X^T = \sigma^2 X X^T = \sigma^2 I$

so $\hat{w}_{LS} = w^* + z'$ where $z' \sim N(0, \sigma^2 I_n)$

$\hat{w}_{top(s)} = T_s(\hat{w}_{LS})$, so $\hat{w}_{top(s)}$ returns the top s entries of the vector $\vec{w}^* + \vec{z'}$

(f) Because $\vec{w}^*$ is s sparsity, and $\hat{w}_{top(s)}$ also only keep s entries and also s sparsity, therefore $\vec{e} = \hat{w}_{top(s)} - \vec{w}^*$ is at most $2s$-sparsity.

(g) $|e_i| = |\hat{w}_{top(s)_i} - w^*_i|$

If $\hat{w}_{top(s)_i} = 0$, $w^*_i = 0$, $|e_i| = 0 \leq 4\sigma\sqrt{\log d}$

If $\hat{w}_{top(s)_i} \neq 0$, $w^*_i = 0$, $|e_i| = |\hat{w}_{top(s)_i}| = |z'_i + w^*_i| = |z'_i| \leq 2\sigma\sqrt{\log d} \leq 4\sigma\sqrt{\log d}$

If $\hat{w}_{top(s)_i} \neq 0$, $w^*_i \neq 0$, $|e_i| = |\hat{w}_{top(s)_i} - w^*_i| = |z'_i| \leq 2\sigma\sqrt{\log d} \leq 4\sigma\sqrt{\log d}$.

If $\hat{w}_{top(s)_i} = 0$, $w^*_i \neq 0$, meaning there is j so that $|w^*_i + z'_i| \leq |z'_j + w^*_j| = |z'_j|$

so $|e_i| = |w^*_i| \leq 2|z_i| = 4\sigma\sqrt{\log d}$.

So we prove that $|e_i| \leq 4\sigma\sqrt{\log d}$

(h) $\|\hat{w}_{top}(s) - w^*\|^2 = \sum_{i=1}^{d} \|e_i\|^2 \le 2s(4\sigma\sqrt{\log d})^2 = 32\sigma^2 s \log d$

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

$= \Pr\{\max_i |e_i| \le 4\sigma\sqrt{\log d}\} = \Pr\{\max(|z_i|) \le 2\sigma\sqrt{\log d}\}$

$\ge 1 - \Pr\{\max_i |z_i| \ge 2\sigma\sqrt{\log d}\}$

$\ge 1 - \frac{1}{d}$

(i) $\frac{1}{n}\|X(\hat{w}_{top}(s) - w^*)\|^2$

$= \frac{1}{n}(X(\hat{w}_{top} - w^*))^T (X(\underbrace{\hat{w}_{top} - w^*}_{\to \vec{a}}))$

$= \frac{1}{n} a^T X^T X a$

$= \frac{1}{n} a^T a$

$= \frac{1}{n}\|\hat{w}_{top}(s) - w^*\|^2 \le \frac{32\sigma^2 s \log d}{n}$

(j) sparsity predictor gives variance of $32\sigma^2 s \log d / n$

and OLS gives variance of $\sigma^2 d / n$

To make the first term smaller than the 2nd term $\Rightarrow 32\sigma^2 s \log d / n \le \sigma^2 d / n$

$\Rightarrow s \le \frac{d}{32 \log d}$

so by leveraging the sparsity assumption, we have smaller variance.

(k) OLS gives variance of $\sigma^2 s / n$

sparse $\hat{w}_{top}(s)$ gives variance of $32\sigma^2 s \log d / n = \frac{\sigma^2 s}{n} \cdot 32 \log d > \frac{\sigma^2 s}{n}$

So when knowing important s features, OLS gives smaller variance compared with sparse $\hat{w}_{top}(s)$ above.

So the price of not knowing important features will introduce larger variance ($\frac{d}{s}$)

4 (a)

```
@staticmethod
def information_gain(X, y, thresh):
    # TODO implement information gain function
    # calculate entropy before cut
    p0 = len(np.where(y==0)[0])/len(y)
    p1 = 1- p0
    entropy_bef = p0 * np.log(p0) + p1 * np.log(p1)

    # calculate entropy after cut
    # calculate for X> thresh
    idx_large = np.where(X>thresh)[0]
    y_large = y[idx_large]
    p0 = len(np.where(y_large==0)[0])/len(y_large)
    p1 = 1- p0
    entropy_aft_large = p0 * np.log(p0) + p1 * np.log(p1)

    idx_small = np.where(X<=thresh)[0]
    y_small = y[idx_small]
    p0 = len(np.where(y_small==0)[0])/len(y_small)
    p1 = 1 - p0
    entropy_aft_small = p0 * np.log(p0) + p1 * np.log(p1)

    # calculate the gain
    w_large = len(idx_large)/len(y)
    w_small = len(idx_small)/len(y)
    gain = entropy_bef - w_large*entropy_aft_large -  w_small*entropy_aft_small
    return gain
```

```
@staticmethod
def gini_impurity(X, y, thresh):
    # TODO implement gini_impurity function
    # calculate G before cut
    p0 = len(np.where(y==0)[0])/len(y)
    p1 = 1- p0
    G_bef = 1 - p0**2 - p1**2

    # calculate G after cut
    # calculate for X> thresh
    idx_large = np.where(X>thresh)[0]
    y_large = y[idx_large]
    p0 = len(np.where(y_large==0)[0])/len(y_large)
    p1 = 1- p0
    G_aft_large = 1- p0**2 - p1**2

    idx_small = np.where(X<=thresh)[0]
    y_small = y[idx_small]
    p0 = len(np.where(y_small==0)[0])/len(y_small)
    p1 = 1 - p0
    G_aft_small = p0 * np.log(p0) + p1 * np.log(p1)

    # calculate the gain
    w_large = len(idx_large)/len(y)
    w_small = len(idx_small)/len(y)
    gain = G_bef - w_large*G_aft_large -  w_small*G_aft_small

    return gain
```

4 (b)

- Miss class labels: ignore this data
- Features are not numerical values: divide into different groups where there are more than 10 data in this group. For each group, assign 1 to the data if it's the same as this group feature; assign 0 if it's different.
- Data miss features: For the hot columns, first assign -1 to it, so it's different than all other features, meaning it's 0 in every group features. For non-hot columns, use the mode of other data for missing features.

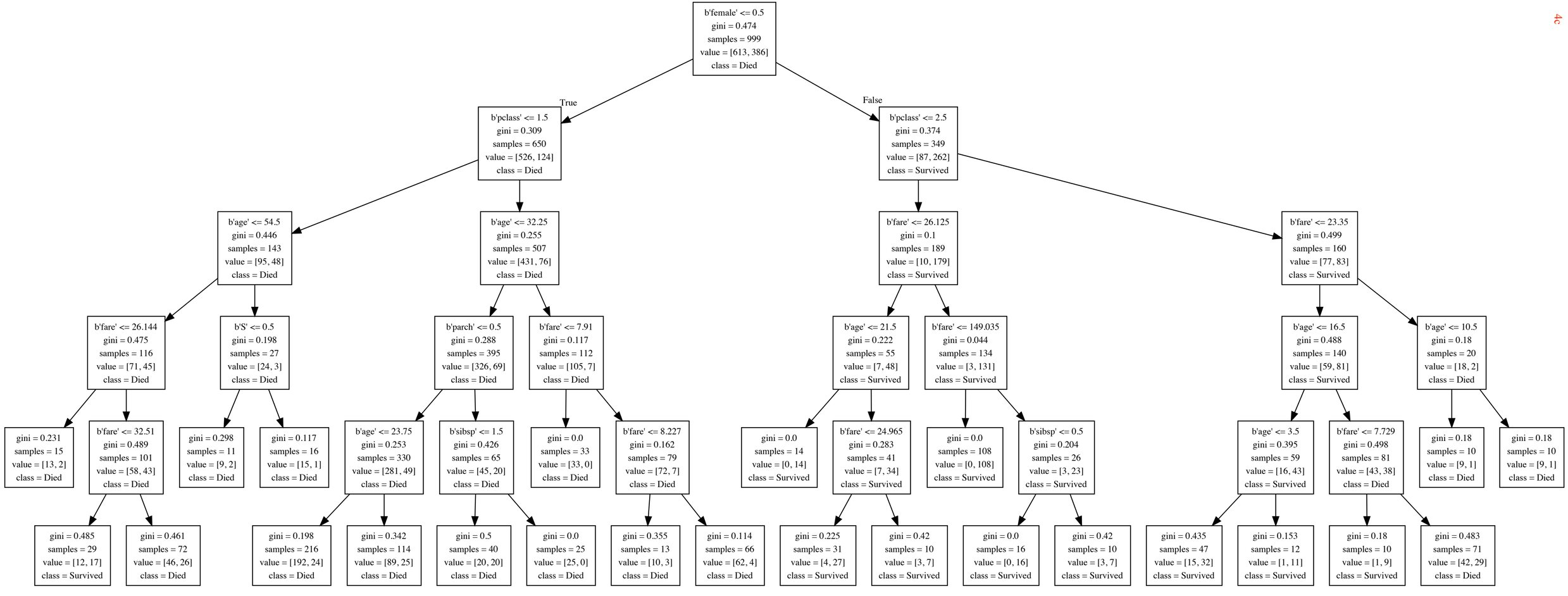
4 (d)

```
class BaggedTrees(BaseEstimator, ClassifierMixin):
    def __init__(self, params=None, n=200):
        if params is None:
            params = {}
        self.params = params
        self.n = n
        self.decision_trees = [
            sklearn.tree.DecisionTreeClassifier(random_state=i, **self.params)
            for i in range(self.n)
        ]

    def fit(self, X, y):
        Ndata = len(y)
        # TODO implement function
        for dt in self.decision_trees:
            idx = np.random.randint(0, Ndata-1, size=Ndata)
            Xi = X[idx]
            yi = y[idx]
            dt.fit(Xi, yi)

    def predict(self, X):
        # TODO implement function
        y = np.zeros(X.shape[0])
        for dt in self.decision_trees:
            y += dt.predict(X)
        y /= self.n
        return np.round(y)
```

4 (e)

For Titanic: The most common splits made at root: First splits [(b'male', 101), (b'female', 99)]
For Spam: First splits [('exclamation', 200)]

```
# TODO implement and evaluate parts c-h
bt = BaggedTrees()
bt.fit(X,y)
evaluate(bt)
```


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

4 (f)

```
class RandomForest(BaggedTrees):
    def __init__(self, params=None, n=200, m=1):
        if params is None:
            params = {}
        self.params = params
        self.n = n
        self.decision_trees = [
            sklearn.tree.DecisionTreeClassifier(random_state=i, max_features=m, **self.params)
            for i in range(self.n)
        ]
        # TODO implement function
        pass
```

4 (g)

```
bt = RandomForest(m=6)
bt.fit(X,y)
evaluate(bt)
```

```
For Titanic: First splits [(b'male', 71), (b'female', 66)
For spam: First splits [('exclamation', 38), ('money', 31)
```

4(h)

```
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
        w = self.w

        # TODO implement function
        # record the number of data points
        ndata = len(y)

        # loop over decision trees
        for i in range(self.n):
            # sample randomly with weights
            idx = np.random.choice(ndata, ndata, p=w)
            Xi = X[idx]
            yi = y[idx]

            # train tree and predict
            self.decision_trees[i].fit(Xi, yi)
            yp = self.decision_trees[i].predict(X)

            # calculate ej and aj
            idx_wrong = np.where(yp!=y)[0]
            ej = np.sum(w[idx_wrong])/np.sum(w)
            aj = 0.5 * np.log((1-ej)/ej)
            self.a[i] = aj

            # update weight
            w = w * np.exp(-aj)
            w[idx_wrong] = w[idx_wrong] * np.exp(aj)

            # normalize w
            w /= np.sum(w)

        self.w = w
        return self

    def predict(self, X):
        # TODO implement function
        # initialize z for c=0/1
        z0 = np.zeros(len(X))
        z1 = np.zeros(len(X))

        # loop over decision trees
        for i in range(self.n):
            yp = self.decision_trees[i].predict(X)

            # calculate z
            idx0 = np.where(yp==0)[0]
            z0[idx0] += self.a[i]
            idx1 = np.where(yp==1)[0]
            z1[idx1] += self.a[i]

        labels = np.zeros(len(X))
        idx = np.where(z0<z1)[0]
        labels[idx] = 1
        return labels
```

Trees are being weighted by a. When trees makes 0 wrong prediction, e=0, corresponding to a=inf ; When trees makes all wrong prediction, e=1, corresponding to a=-inf. So we give more weights to trees which makes more correct predictions.

This algorithm gives more weights to data that are not predicted correctly. At each step, if a data is not predicted correctly, it will be given weights of exp(aj); On the contrary, if a data is predicted correctly, it will be given weights of exp(-aj). So next time, those data who are not predicted correctly will have more probability when sampling.

ai<0 means ej>0.5, meaning this tree has large errors. but ada boots always assume ej<0.5, so those trees will flip their prediction.

4(i)
For titanic: First splits [(b'fare', 42), (b'age', 37)
For spam: First splits [('meter', 32), ('volumes', 28)

The data that is hardest to predict is with large weight, like:

```
[ 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.]
[ 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.  2.  0.  0.]
meaning they has no other keys except &  or (
```

The data that is easiest to predict is with small weight, like:

```
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
0. 1.  4.  0.  0.  0.  0.  1.  0.  0.  0.  0.  2.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.
0. 0.  3.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
meaning they have “meter” in it
```

```
4 (j)
For titanic:
Single tree
Cross validation [ 0.79640719  0.76576577  0.79819277]
accuracy: 0.822823

Bagged trees
Cross validation [ 0.79640719  0.75675676  0.78915663]
accuracy: 0.973974

Random forests
Cross validation [ 0.79041916  0.75675676  0.79216867]
accuracy: 0.973974

Ada Boost
Cross validation [ 0.80538922  0.74774775  0.77108434]
accuracy: 0.973974
```

For spam
Single tree
Cross validation [0.80104408 0.80800464 0.77900232]
accuracy: 0.809938

Bagged trees
Cross validation [0.82192575 0.81786543 0.78016241]
accuracy: 0.885924

Random forests
Cross validation [0.82134571 0.81612529 0.78132251]
accuracy: 0.885924

Ada Boost
Cross validation [0.82656613 0.82424594 0.78828306]
accuracy: 0.885924

See submission for test data prediction

(k)
see submission

5. Q: In the m-th step of Adaboost, we want to minimize

$$\arg\min_{\alpha_m, G_m} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \alpha_m G_m(x_i))$$

in which L is exponential loss: $L(y, h(x)) = e^{-yh(x)}$

derive the expression for α_m.

A:

$$\alpha_m, G_m = \arg\min_{\alpha, G} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \alpha G(x_i))$$

Let $w_m^i = e^{-y_i F_{m-1}(x_i)}$

$$\alpha_m, G_m = \arg\min_{\alpha, G} \sum w_m^i e^{-y_i \alpha G(x_i)}$$

$$= \arg\min_{\alpha, G} \left(\sum_{y_i = G(x_i)} w_m^i e^{-\alpha} + \sum_{y_i \neq G(x_i)} w_m^i e^{\alpha} \right)$$

$$= \arg\min_{\alpha, G} \sum_{i=1}^{n} w_m^i e^{-\alpha} + \sum_{y_i \neq G(x_i)} w_m^i (e^{\alpha} - e^{-\alpha})$$

$$= \sum_{i=1}^{n} w_m^i \arg\min_{\alpha, G} e^{-\alpha} + \frac{\sum_{y_i \neq G(x_i)} w_m^i}{\sum w_m^i} (e^{\alpha} - e^{-\alpha})$$

Let $e_m = \dfrac{\sum_{y_i \neq G(x_i)} w_m^i}{\sum w_m^i}$

$$\alpha_m, G_m = \arg\min_{\alpha, G} e^{-\alpha} + e_m (e^{\alpha} - e^{-\alpha}) = F(\alpha)$$

$$\frac{\partial F(\alpha)}{\partial \alpha} = 0 \Rightarrow \alpha = \frac{1}{2} \ln \left(\frac{1 - e_m}{e_m} \right)$$