HW2-1

a)

Names	Email Address
Wan	Jhun0324@berkeley.edu

Description of Team: Best Group Ever

How did I work?

Comments:

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.

Qingyang Zhao

HW12-2

a)

$$J_\lambda(w) = \frac{1}{2}\|y - Aw\|_2^2 + \lambda\|w\|_1 \quad , \lambda > 0$$

$$= \frac{1}{2}(y-Aw)^T(y-Aw) + \lambda\sum_i |w_i| \quad , \lambda > 0$$

$$= \frac{1}{2}(y^Ty - 2w^TA^Ty + w^TA^TAw) + \lambda\sum_i |w_i|$$

$$= \frac{1}{2}(y^Ty - 2\sum_i w_i A_i^T y + n\sum_i w_i^2) + \lambda\sum_i |w_i|$$

Thus for each w_i^*, only related with A_i, which means the ith feature determines w_i^* regardless other feature.

b)

$w_i^* > 0$

$$\frac{\partial J_\lambda(w)}{\partial w_i} = -A_i^Ty + nw_i + \lambda$$

$$w_i = \frac{A_i^Ty - \lambda}{n}$$

c)

$w_i^* < 0$

$$\frac{\partial J_\lambda(w)}{\partial w_i} = -A_i^Ty + nw_i - \lambda$$

$$w_i = \frac{A_i^Ty + \lambda}{n}$$

d)

in terms of w_i, our objective: $F(w_i) = -w_iA_i^Ty + \frac{1}{2}nw_i^2 + \lambda|w_i|$

when $w_i = 0$ objective $= 0$

Thus when $F(w_i)\big|_{w_i = 0}$

$$F(w_i)\Big|_{w_i = \frac{A_i^Ty - \lambda}{n}} = -\frac{(A_i^Ty)^2 - \lambda A_i^Ty}{n} + \frac{(A_i^Ty)^2 - 2\lambda A_i^Ty + \lambda^2}{2n} + \frac{\lambda A_i^Ty - \lambda^2}{n}$$

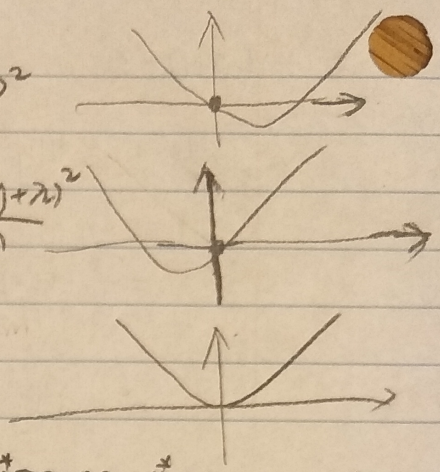
$$= -\frac{(A_i^Ty - \lambda)^2}{2n}$$

$$F(w_i)\Big|_{w_i = \frac{A_i^Ty + \lambda}{n}} = -\frac{(A_i^Ty)^2 + \lambda(A_i^Ty)}{n} + \frac{(A_i^Ty)^2 + \lambda^2 + 2A_i^Ty\lambda}{2n} + \frac{A_i^Ty\lambda + \lambda^2}{n} = \frac{(A_i^Ty + \lambda)^2}{2n}$$

Thus $w_i > 0$, $w_i = \frac{A_i^T y - \lambda}{n}$ $\quad F(w_i^*) = -\frac{(A_i^T y - \lambda)^2}{2n}$

$w_i < 0$, $w_i = \frac{A_i^T y + \lambda}{n}$ $\quad F(w_i^*) = \frac{-(A_i^T y + \lambda)^2}{2n}$

$w_i = 0$ $\quad F(w_i^*) = 0$



when $\frac{A_i^T y - \lambda}{n} \le 0$ and $\frac{A_i^T y + \lambda}{n} \ge 0$ neither $w_i^* > 0$ or $w_i^* < 0 \rightarrow w_i^* = 0$

$\rightarrow \quad -\lambda \le A_i^T y \le \lambda \;,\; w_i^* = 0$

e) $J_{\lambda_2}(w) = \frac{1}{2}\|y - Aw\|_2^2 + \lambda \|w\|_2^2$

$J_{\lambda_2}(w) = \frac{1}{2}(y^T y - 2w^T A^T y + w^T A^T A w) + \lambda w^T w$

$= \frac{1}{2}(y^T y - 2\sum_i w_i A_i^T y + n\sum_i w_i^2) + \lambda \sum_i w_i^2$

$\frac{\partial J_{\lambda_2}(w)}{\partial w_i} = -A_i^T y + n w_i + 2\lambda w_i = 0.$

$w_i = \frac{A_i^T y}{n + 2\lambda}$ $\quad$ when $A_i^T y = 0$ $\quad w_i = 0$

For LASSO, the condition is a range
But for L-2 norm its a value.

Thus LASSO gives Sparsity.

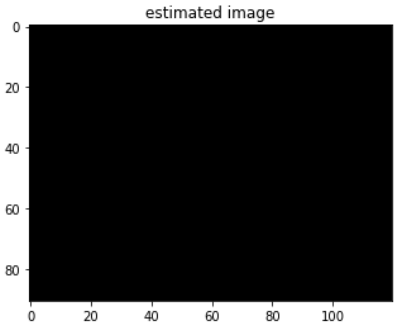
f) see pdf next

HW12 – 2

f) Because The condition for generating sparsity(w_i = 0) is that A_i^T*y within [-lambda, lambda] This means when lambda is too large, there will be so many zeros.

```
# lambda is large
a = 1
recovered = LASSO((height,width),measurements,A,a)
```

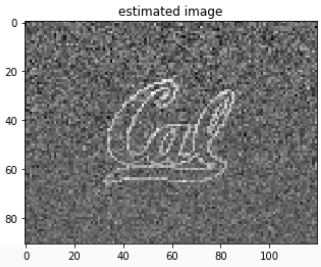
```
[[ 0. -0. -0. ...,  0.  0. -0.]
 [-0.  0. -0. ..., -0.  0.  0.]
 [ 0. -0.  0. ...,  0.  0. -0.]
 ...,
 [-0.  0. -0. ...,  0. -0.  0.]
 [-0. -0. -0. ..., -0. -0. -0.]
 [ 0.  0.  0. ...,  0.  0. -0.]]
```



Because the condition for generating sparsity(w_i = 0) is that A_i^T*y within [-lambda, lambda] This means when lambda is too small, there will be less zeros.

```
# lambda is small
a = 0.000000001
recovered = LASSO((height,width),measurements,A,a)
```

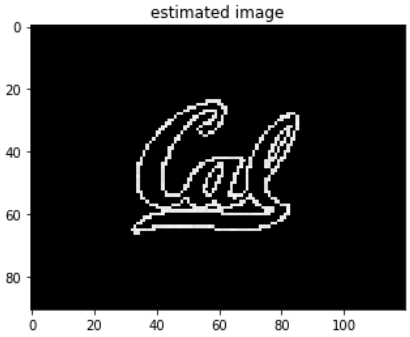
```
[[-0.03662078 -0.45207691 -0.07083422 ...,  0.0758507  -0.07009895
  -0.0688969 ]
 [-0.50529875  0.16315041 -0.15226952 ..., -0.00080187  0.1945343
   0.08630258]
 [-0.02972023 -0.15762112  0.0226709  ...,  0.21082238  0.16025736
  -0.72464153]
 ...,
 [-0.0400522  -0.02515765 -0.04686682 ..., -0.19826164 -0.14973695
  -0.10333221]
 [ 0.20370217 -0.02463526 -0.10273393 ...,  0.07001685  0.07400616
   0.01256854]
 [ 0.04517431 -0.21560573  0.07605233 ...,  0.06980866  0.00223421
  -0.02412968]]
```



Thus a suitable lambda is the key to generate sparsity.

```
# appropriate lambda
a = 0.00001
recovered = LASSO((height,width),measurements,A,a)
```

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



HW12-3

a)

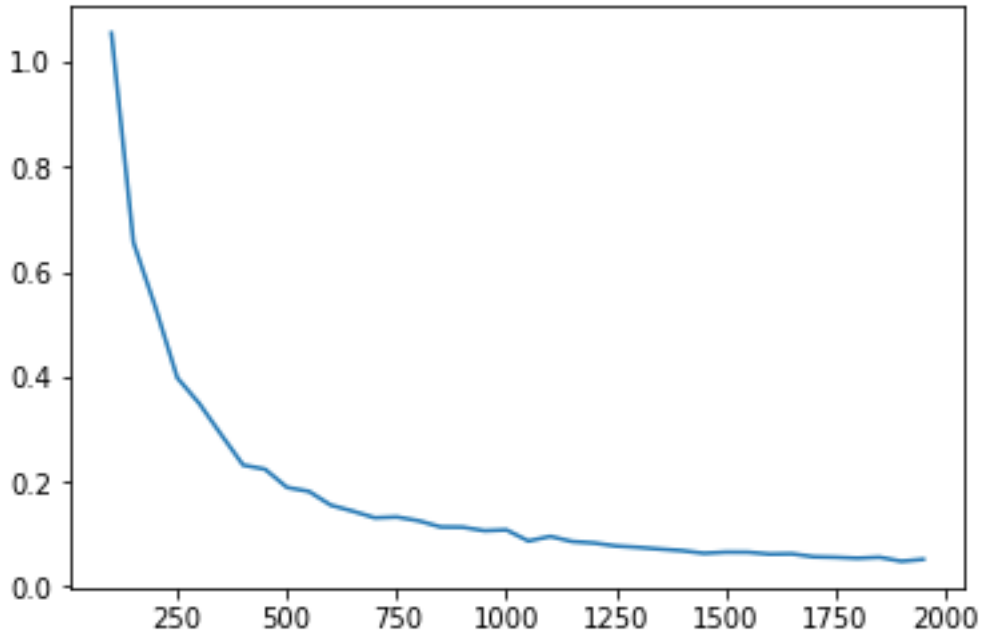
As sample points increase, error will decrease.

As feature dimension increases, erroe will increase because of model complexity.

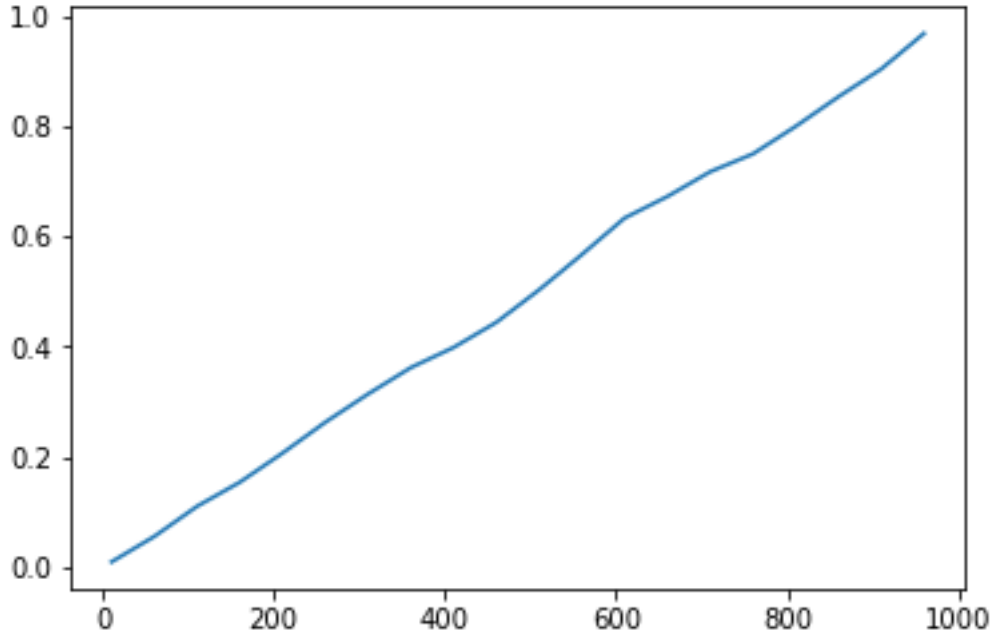
As spasity increase, it will first decrease then increse, because appropriate number of features should be decided. Since there is tradeoff between the model complexity and feature utility.

para: srange, ls_error_s, _, _ = error_calc(num_iters=10, param='s', n=1000, d=100, s=5, s_model=True, true_s=5)

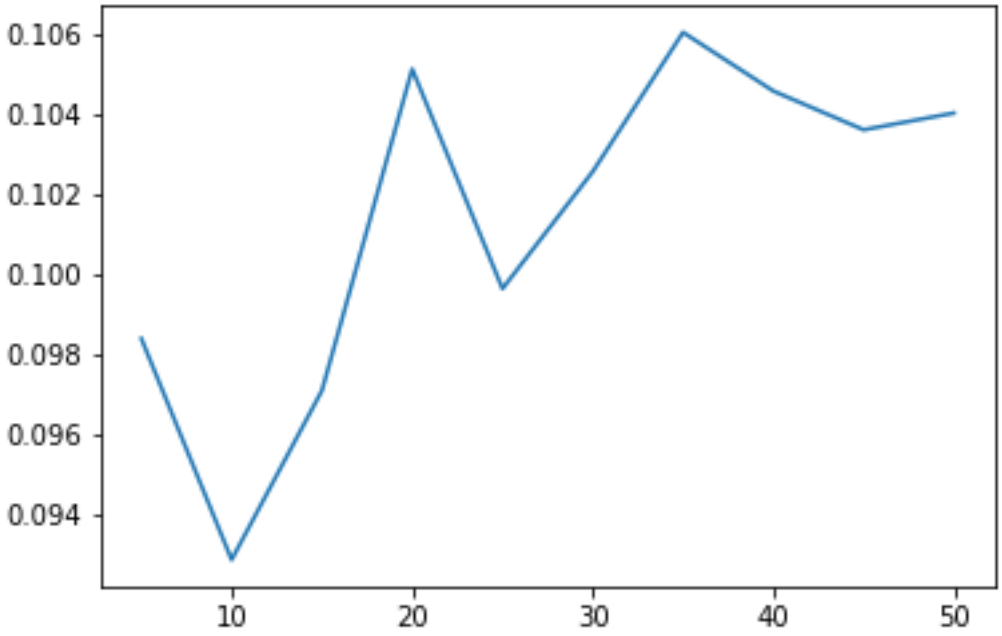
n-error:



d-error:



s-error:



HW12-3

d) Since: $Pr(|z| \geq t) \leq e^{-\frac{t^2}{2\sigma^2}}$

let $t = 2\sigma\sqrt{\log d}$

$$Pr(|z| \geq 2\sigma\sqrt{\log d}) \leq (\frac{1}{d})^2$$

$$Pr(\max_{i \in \{1 \ldots d\}} |z_i| \geq t) \leq Pr(\bigcup_i |z_i| \geq t) \leq \sum_i Pr(|z_i| \geq t) = (\frac{1}{d})^2 \times d = \frac{1}{d}$$

$$\Rightarrow Pr\{\max_{i \in \{1,2 \ldots d\}} |z_i| \geq 2\sigma\sqrt{\log d}\} \leq \frac{1}{d}$$

e) zero out all but the top k entries of $\hat{w}_{LS}$

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

$= T_s(w^* + z')$

which means $\hat{w}_{top(s)}$ is the first k entries of $w^* + z'$ in absolute value

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

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

$\hat{w}_{LS} = (A^TA)^{-1}A^Ty$

$= (A^TA)^{-1}A^T(Aw^* + z)$

$= (A^TA)^{-1}A^TAw^* + (A^TA)^{-1}A^Tz$

$= w^* + A^Tz$ since A^T orthogonal columns, A^Tz is also $\sim (0, \sigma^2)$

$= w^* + z'$ (i.e. $A^Tz = z'$)

f) $e = \hat{w}_{top(s)} - w^*$

$T_s(|w^* + z'|) - w^* = \begin{cases} T_s(z_i')\ w_i^* \geq -z_i' \\ T_s(-z_i' - 2w_i^*)\ w_i^* < -z_i' \end{cases}$

[0 ... 0 ... 0] (s, d-s)

g) $S = \{\max |z_i'| \leq \sigma\sqrt{\log d}\}$

$|e_i| = ||w_i^* + z_i'| - w_i^*| = \begin{cases} |z_i'| & w_i^* + z_i' \geq 0 \\ |-z_i' - 2w_i^*| & w_i^* + z_i' < 0 \end{cases}$

$\leq |z_i'| + |2w_i^*|$

HW12- 4

a)

Entropy and information gain:

I(X;Y) = H(Y) – H (Y|X)

Algo Code:

```
    @staticmethod
    def entropy(y):
        # TODO implement entropy function
        uni_y = np.unique(y)
        prob_y = []
        for i in range(np.size(uni_y)):
            prob_y.append(list(y).count(uni_y[i])/np.size(y))
        return - np.dot(np.asarray(prob_y),np.log(np.asarray(prob_y)))

    @staticmethod
    def information_gain(X, y, thresh):
        # TODO implement information gain function
        # X0, y0, X1, y1 = self.split(X, y, 0, thresh)
        # X0, y0, X1, y1 = DecisionTree.split(X = X, y = y, idx = 0, thresh=thresh)
        idx0 = np.where(X[:] < thresh)[0]
        idx1 = np.where(X[:] >= thresh)[0]
        X0, X1 = X[idx0], X[idx1]
        y0, y1 = y[idx0], y[idx1]

        prob_X0 = float(np.size(X0))/(np.size(X0) + np.size(X1))
        prob_X1 = float(np.size(X1))/(np.size(X0) + np.size(X1))
        return DecisionTree.entropy(y) - (prob_X0*DecisionTree.entropy(y0) +
prob_X1*DecisionTree.entropy(y1))
```

Main **Code:**

```
# Basic decision tree
print("\n\nPart (a-b): simplified decision tree\n")

kf = KFold(n_splits=3, shuffle=True)

for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index, :], X[test_index, :]
    y_train, y_test = y[train_index], y[test_index]

    dt = DecisionTree(max_depth = 3, feature_labels=features)
    dt.fit(X_train, y_train)
    print(dt.split_idx, dt.thresh)
    print('training accuracy :')
    print(np.sum(dt.predict(X_train) == y_train)/y_train.shape[0])
    print('validation Accuracy :')
    print(np.sum(dt.predict(X_test) == y_test)/y_test.shape[0],'\n')
```

b) preprocessing:

1.using one-hot encoding:
pclass, sex, sibsp, parch, embarked

2.treat as continuous:
age:

1) age has too many values, one hot encoding will increase model complexity.
2) The order makes sense.

Fare: continuous itself

3.Delete feature:
Ticket, cabin

Missing value: nan->0

Accuracy:

{"Previous Highest Score": 100.0, "Used Daily Submissions": 0, "Score": 100.0, "Accuracy": 0.7580645161290323, "Highest Score": 100.0}

code:

```
        if dataset == "titanic":
            # Load titanic data
            # path_train = 'datasets/titanic/titanic_training.csv'
            path_train = 'titanic_training.csv'
            data = genfromtxt(path_train, delimiter=',', dtype=np.float32)
            features = data[0, 1:]  # features = all columns except survived
            y = data[1:, 0]  # label = survived

            #change the original data
            data[np.isnan(data)]=0
            # class_names = ["Died", "Survived"]

            # TODO implement preprocessing of Titanic dataset
            X_pre =
np.asarray([data[1:,1],data[1:,2],data[1:,4],data[1:,5],data[1:,7],data[1:,3],data[1:,6]])
            a = OneHotEncoder(categorical_features = np.array([0,1,2,3,4]))
            a.fit(X_pre.T)
            X = a.transform(X_pre.T).toarray()

            path_test = 'titanic_testing_data.csv'
            data_test = genfromtxt(path_test, delimiter=',', dtype=np.float32)
            data_test[np.isnan(data_test)]=0
            Z_pre =
np.asarray([data_test[1:,0],data_test[1:,1],data_test[1:,3],data_test[1:,4],data_test[1:,6
],data_test[1:,2],data_test[1:,5]])
            Z = a.transform(Z_pre.T).toarray()
```

c)
Titanic:

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

4 1e-05
training accuracy :
0.804804804805
validation Accuracy :
0.784431137725

4 1e-05
training accuracy :
0.799100449775
validation Accuracy :
0.804804804805

4 1e-05
training accuracy :
0.7976011994
validation Accuracy :
0.795795795796
```

Spam:

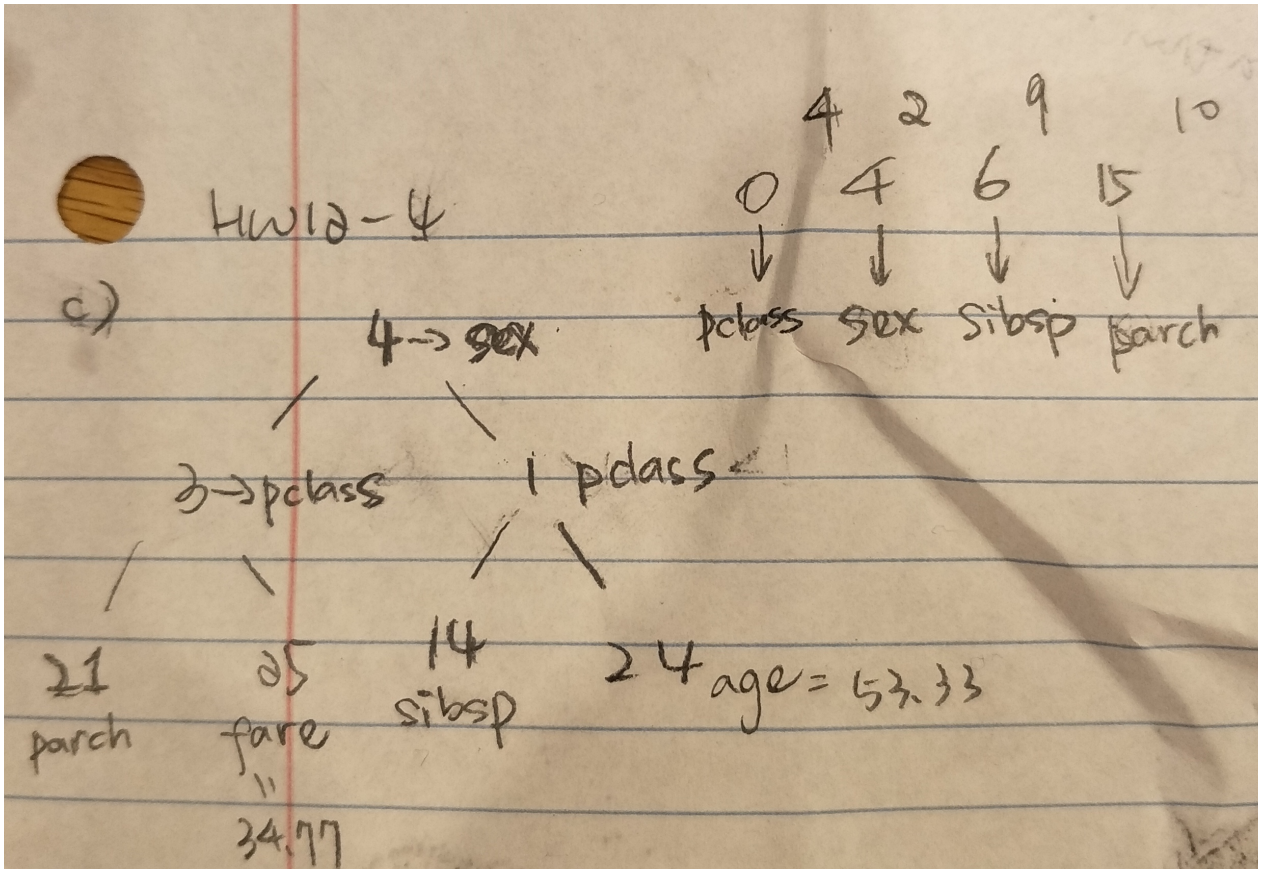
```
Part (a-b): simplified decision tree

28 1e-05
training accuracy :
0.794953596288
validation Accuracy :
0.794663573086

28 1e-05
training accuracy :
0.794663573086
validation Accuracy :
0.79524361949

28 1e-05
training accuracy :
0.795533642691
validation Accuracy :
0.794083526682
```

Visualize for titanic:


HW10-4
4 2 9 10
0 4 6 15
pclass sex sibsp parch
c)
4 -> sex
3 -> pclass
1 pclass
21
parch
25
fare
"
34.77
14
sibsp
24
age = 53.33

d)
Bagged Trees Code:
Algo Code:

```
class BaggedTrees(BaseEstimator, ClassifierMixin):

    def __init__(self, params=None, n=200, sample_percent = 0.2):
        if params is None:
            params = {}
        self.params = params
        self.n = n
        self.sample_percent = sample_percent
        self.decision_trees = [
            DecisionTreeClassifier(random_state=i, **self.params) for i in
            range(self.n)]
        self.accu = []
        self.bestAccu = 0
        self.bestTreeInd = None

    def fit(self, X, y):
        # TODO implement function
        for j in range(self.n):
            sub_ind = np.random.choice(range(X.shape[0]),
int(self.sample_percent*X.shape[0]), replace = True)
            X_new = X[sub_ind, :]
            y_new = y[sub_ind]
            for i in range(int(1/self.sample_percent)):
                sub_ind = np.random.choice(range(X.shape[0]),
int(self.sample_percent*X.shape[0]), replace = True)
                X_new = np.concatenate((X_new, X[sub_ind, :]),axis = 0)
                y_new = np.concatenate((y_new, y[sub_ind]),axis = 0)
            self.decision_trees[j].fit(X_new, y_new)
            self.accu.append(np.sum(self.decision_trees[j].predict(X) == y)/y.shape[0])
        self.bestAccu = np.max(self.accu)
        self.bestTreeInd = np.argmax(self.accu)
        return self


    def predict(self, X):
        # TODO implement function
        return self.decision_trees[self.bestTreeInd].predict(X)
```

Main **Code:**

```
    # d part bagged tree
    print("\n\nPart (d): Bagged decision tree\n")

    kf = KFold(n_splits=3, shuffle=True)

    for train_index, test_index in kf.split(X):
        dt = BaggedTrees(params=params,sample_percent = 0.3)
        dt.fit(X_train, y_train)

        # calculate common split on the root
        feature_split = []
        for i in range(dt.n):

feature_split.append((dt.decision_trees[i].tree_.feature[0],dt.decision_trees[i].tree_.thr
eshold[0]))
        result = dict((i, feature_split.count(i)) for i in feature_split)
        maximum = max(result, key=result.get)
        print(maximum, result[maximum])

        print('training Accuracy :')
        print(dt.bestAccu)
        print('validation Accuracy :')
        print(np.sum(dt.predict(X_test) == y_test)/y_test.shape[0],'\n')
```

e)
depth = 5
using bagged trees the performance is better,
the test error on titanic is: (2% better than simple decision tree)

{"Score": 100.0, "Previous Highest Score": 100.0, "Used Daily Submissions": 4, "Highest Score": 100.0, "Accuracy": 0.7774193548387097}

the common split on ‘titanic’ is
feature 5, threshold = 0.5, which represents for **sex** attribute. (108 trees)

```
Part (d): Bagged decision tree

(5, 0.5) 108
training Accuracy :
0.84107946027
validation Accuracy :
0.795795795796

(5, 0.5) 110
training Accuracy :
0.833583208396
validation Accuracy :
0.792792792793

(5, 0.5) 109
training Accuracy :
0.84107946027
validation Accuracy :
0.777777777778
```

the common split on ‘spam’ is

28th feature represents for ‘#’ (200 trees)

```
Part (d): Bagged decision tree

(28, 0.5) 200
training Accuracy :
0.816995359629
validation Accuracy :
0.808584686775

(28, 0.5) 200
training Accuracy :
0.816995359629
validation Accuracy :
0.81090487239

(28, 0.5) 200
training Accuracy :
0.815835266821
validation Accuracy :
0.803944315545
```

f)
implementation for random forest:

Algo Code:

```
class RandomForest(BaggedTrees):

    def __init__(self, params=None, n=200, m=1):
        if params is None:
            params = {}
        # TODO implement function
        self.params = params
        self.n = n
        #self.sample_percent = sample_percent
        self.decision_trees = [
            DecisionTreeClassifier(random_state=i, **self.params) for i in
            range(self.n)]
        self.accu = []
        self.bestAccu = 0
        self.bestTreeInd = None
        self.m = m
        self.featInd = []

    def fit(self, X, y):
        for j in range(self.n):
            sub_ind = np.random.choice(range(X.shape[0]), X.shape[0], replace = True)
            sub_ind_feat = np.random.choice(range(X.shape[1]), self.m, replace = True)
            X_temp = X[sub_ind, :]
            X_new = X_temp[:, sub_ind_feat]
            y_new = y[sub_ind]
            # for i in range(int(1/self.sample_percent)):
            '''for i in range(X.shape[0]-1):
                sub_ind = np.random.choice(range(X.shape[0]), 1, replace = True)
                # sub_ind = np.random.choice(range(X.shape[0]), int(self.sample_percent*X.shape[0]), replace =
                X_temp = X[sub_ind, :]
                X_new = np.concatenate((X_new, X_temp[:, sub_ind_feat]),axis = 0)
                y_new = np.concatenate((y_new, y[sub_ind]),axis = 0)'''
            self.featInd.append(sub_ind_feat)
            self.decision_trees[j].fit(X_new, y_new)
            X_accur = X[:, sub_ind_feat]
            self.accu.append(np.sum(self.decision_trees[j].predict(X_accur) == y)/y.shape[0])
        self.bestAccu = np.max(self.accu)
        self.bestTreeInd = np.argmax(self.accu)

        return self

    def predict(self, X):
        # TODO implement function
        X_new = X[:, self.featInd[self.bestTreeInd]]
        return self.decision_trees[self.bestTreeInd].predict(X_new)
```

*Main* Code:

```
    # f) random forest
    print("\n\nPart (f): Random Forest decision tree\n")

    kf = KFold(n_splits=3, shuffle=True)

    for train_index, test_index in kf.split(X):
        dt = RandomForest(params=params, n=200, m=15)
        dt.fit(X_train, y_train)


        feature_split = []
        for i in range(dt.n):

feature_split.append((dt.decision_trees[i].tree_.feature[0],dt.decision_trees[i].tree_.thr
eshold[0]))
        result = dict((i, feature_split.count(i)) for i in feature_split)
        maximum = max(result, key=result.get)
        print(maximum, result[maximum])

        print('training Accuracy:')
        print(dt.bestAccu)
        print('validation Accuracy:')
        print(np.sum(dt.predict(X_test) == y_test)/y_test.shape[0],'\n')


    # np.savetxt('submission.txt',dt.predict(Z)[:].astype(np.int8))
```

g) depth = 5, m = 15

the common split on ‘titanic’ is (first data)
feature 13, threshold = 0.5, which represents for **sibling** attribute. (17 trees)

for titanic:

```
Part (f): Random Forest decision tree

(13, 0.5) 17
training Accuracy:
0.826086956522
validation Accuracy:
0.783783783784

(11, 0.5) 20
training Accuracy:
0.826086956522
validation Accuracy:
0.78978978979

(2, 0.5) 18
training Accuracy:
0.830584707646
validation Accuracy:
0.792792792793
```

for spam data:
the common split on ‘spam’ is
9th feature represents for ‘**featured’** (19 trees)

```
Part (f): Random Forest decision tree

(9, 0.5) 19
training Accuracy:
0.812645011601
validation Accuracy:
0.806844547564

(4, 0.5) 20
training Accuracy:
0.810614849188
validation Accuracy:
0.79060324826

(2, 0.5) 21
training Accuracy:
0.810324825986
validation Accuracy:
0.805684454756
```

h)

Algo Code:

```
class BoostedRandomForest(RandomForest):

    def __init__(self, params = None, M_tree = 200, m = 2):
        if params is None:
            params = {}
        # TODO implement function
        self.params = params
        self.M = M_tree
        self.m = m
        # self.decision_trees = DecisionTreeClassifier(random_state=0, **self.params)
        self.decision_trees = [
            DecisionTreeClassifier(random_state=i, **self.params) for i in
            range(self.M)]
        self.featInd = []


    def fit(self, X, y):
        self.w = np.ones(X.shape[0]) / X.shape[0]  # Weights on data
        self.a = np.zeros(self.M)  # Weights on decision trees

        # TODO implement function
        for i in range(self.M):
            # generate samples using weight
            sub_ind = np.random.choice(range(X.shape[0]), X.shape[0], replace = True, p = self.w/np.sum(self.w))
            sub_ind_feat = np.random.choice(range(X.shape[1]), self.m, replace = True)
            self.featInd.append(sub_ind_feat)
            X_temp = X[sub_ind, :]
            X_new = X_temp[:, sub_ind_feat]
            y_new = y[sub_ind]
            self.decision_trees[i].fit(X_new, y_new)
            X_test = X[:, sub_ind_feat]
            e = np.dot(self.decision_trees[i].predict(X_test) == y, self.w)/np.sum(self.w)
            self.a[i] = np.log((1-e)/e)/2
            # print(e)
            ind_0 = np.where((self.decision_trees[i].predict(X_test) == y) == False)
            ind_1 = np.where((self.decision_trees[i].predict(X_test) == y) == True)
            self.w[ind_0] = self.w[ind_0] * np.exp(-self.a[i])
            self.w[ind_1] = self.w[ind_1] * np.exp(self.a[i])

        return self

    def predict(self, X):
        # TODO implement function
        pred_0 = np.zeros(X.shape[0])
        pred_1 = np.zeros(X.shape[0])
        for i in range(self.m):
            pred_0 = pred_0 + self.a[i]*(self.decision_trees[i].predict(X[:,self.featInd[i]]) == np.zeros(X.shape[0]))
            pred_1 = pred_1 + self.a[i]*(self.decision_trees[i].predict(X[:,self.featInd[i]]) == np.ones(X.shape[0]))


        temp = pred_0 - pred_1
        temp[temp >= 0] = 1
        temp[temp < 0] = 0
        return temp

        #pass
```

Main Code:

```
    # h) Boosted random forest
    print("\n\nPart (h): Boosted Random Forest decision tree\n")

    kf = KFold(n_splits=3, shuffle=True)

    for train_index, test_index in kf.split(X):
        dt = BoostedRandomForest(params=params, M_tree = 100, m = 20)
        dt.fit(X_train, y_train)

        # calculate common split on the root
        feature_split = []
        for i in range(dt.M):

feature_split.append((dt.decision_trees[i].tree_.feature[0],dt.decision_trees[i].tree_.thr
eshold[0]))
        result = dict((i, feature_split.count(i)) for i in feature_split)
        maximum = max(result, key=result.get)
        print(maximum, result[maximum])

        print('training Accuracy:')
        print(np.sum(dt.predict(X_train) == y_train)/y_train.shape[0])
        print('validation Accuracy:')
        print(np.sum(dt.predict(X_test) == y_test)/y_test.shape[0],'\n')
```

i) Depth = 5 , m = 10, M = 8

Titanic:

the common split on ‘titanic’ is (first data)

feature 6, threshold = 0.5, which represents for **sibling** attribute. (1 trees)

```
Part (h): Boosted Random Forest decision tree

(6, 75.245849609375) 1
training Accuracy:
0.823088455772
validation Accuracy:
0.786786786787

(7, 0.5) 2
training Accuracy:
0.817091454273
validation Accuracy:
0.780780780781

(7, 0.5) 3
training Accuracy:
0.806596701649
validation Accuracy:
0.804804804805
```

Spam:

for spam data:

the common split on ‘spam’ is

10th feature represents for ‘**differ’** (19 trees)

```
Part (h): Boosted Random Forest decision tree

(10, 0.5) 9
training Accuracy:
0.837587006961
validation Accuracy:
0.820185614849

(16, 0.5) 8
training Accuracy:
0.831496519722
validation Accuracy:
0.813225058005

(19, 0.5) 8
training Accuracy:
0.840487238979
validation Accuracy:
0.810324825986
```

j)
Summary:
Stability:
single tree > bagged tree > random forest > boosted
Other Benefits:
It really depends, single tree/ bagged tree has more model complexity. And longer run time.
While random forest and boosted algorithm has less complexity and running time,
But we should use cross validation to choose hyper-parameter, and the randomness may result in various accuracy.

Spam:

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

28 1e-05
training accuracy :
0.794953596288
validation Accuracy :
0.794663573086

28 1e-05
training accuracy :
0.794663573086
validation Accuracy :
0.79524361949

28 1e-05
training accuracy :
0.795533642691
validation Accuracy :
0.794083526682



Part (d): Bagged decision tree

(28, 0.5) 200
training Accuracy :
0.816995359629
validation Accuracy :
0.808584686775

(28, 0.5) 200
training Accuracy :
0.816995359629
validation Accuracy :
0.81090487239

(28, 0.5) 200
training Accuracy :
0.815835266821
validation Accuracy :
0.803944315545



Part (f): Random Forest decision tree

(9, 0.5) 19
training Accuracy:
0.812645011601
validation Accuracy:
0.806844547564

(4, 0.5) 20
training Accuracy:
0.810614849188
validation Accuracy:
0.79060324826

(2, 0.5) 21
training Accuracy:
0.810324825986
validation Accuracy:
0.805684454756
```

```
Part (h): Boosted Random Forest decision tree

(10, 0.5) 9
training Accuracy:
0.837587006961
validation Accuracy:
0.820185614849

(16, 0.5) 8
training Accuracy:
0.831496519722
validation Accuracy:
0.813225058005

(19, 0.5) 8
training Accuracy:
0.840487238979
validation Accuracy:
0.810324825986
```

Titanic:

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

4 1e-05
training accuracy :
0.804804804805
validation Accuracy :
0.784431137725

4 1e-05
training accuracy :
0.799100449775
validation Accuracy :
0.804804804805

4 1e-05
training accuracy :
0.7976011994
validation Accuracy :
0.795795795796



Part (d): Bagged decision tree

(5, 0.5) 108
training Accuracy :
0.84107946027
validation Accuracy :
0.795795795796

(5, 0.5) 110
training Accuracy :
0.833583208396
validation Accuracy :
0.792792792793

(5, 0.5) 109
training Accuracy :
0.84107946027
validation Accuracy :
0.777777777778



Part (f): Random Forest decision tree

(13, 0.5) 17
training Accuracy:
0.826086956522
validation Accuracy:
0.783783783784

(11, 0.5) 20
training Accuracy:
0.826086956522
validation Accuracy:
0.78978978979

(2, 0.5) 18
training Accuracy:
0.830584707646
validation Accuracy:
```

```
Part (h): Boosted Random Forest decision tree

(6, 75.245849609375) 1
training Accuracy:
0.823088455772
validation Accuracy:
0.786786786787

(7, 0.5) 2
training Accuracy:
0.817091454273
validation Accuracy:
0.780780780781

(7, 0.5) 3
training Accuracy:
0.806596701649
validation Accuracy:
0.804804804805
```