HW5

Name: Di Zhen

ID: 3035272293

1 Decision Trees for Classification

1.1 Implement Decision Trees

In [1]:

```
class DecisionTree:

    class Node:
        def __init__(self, split_rule, left, right, label, is_leaf):
            self.split_rule = split_rule
            self.left = left
            self.right = right
            self.label = label
            self.is_leaf = is_leaf # 1 = stop

        def __repr__(self):
            """
            TODO: one way to visualize the decision tree is to write out a __repr__ met
hod
            that returns the string representation of a tree. Think about how to visual
ize
            a tree structure. You might have seen this before in CS61A.
            """
            def viz(Node, prefix, symbol):
                if not Node:
                    return prefix + '[]'
                if Node.is_leaf == 1:
                    return(prefix + '(Therefore the email was: '+ class_names[Node.labe
l] + ')')
                else:
                    ret = (prefix + '[Feature: '+ features[Node.split_rule[0]] +
                           ', Threshold: '+symbol+ str(Node.split_rule[1]) + ']')
                    ret += '\n' + viz(Node.left, prefix + '\t','<=') + '\n' + viz(Node.
right, prefix + '\t','>')
                    return ret

            return viz(self, "",'<=')


    def __init__(self, max_depth = 200):
        self.max_depth = max_depth

    def max_count(self, array):
        return stats.mode(array, nan_policy='omit')[0][0]


    def entropy(self,y):
        p = y / (np.sum(y)+1e-10)
        return -p.dot(np.log2(p+1e-10))


    def entropy_impurity(self,left_y_freq, right_y_freq):
        Sl = np.sum(left_y_freq)
        Sr = np.sum(right_y_freq)
        return (Sl * self.entropy(left_y_freq) + Sr * self.entropy(right_y_freq)) / (Sl
+Sr)


    def information_gain(self,left_y_freq, right_y_freq):
        total = left_y_freq + right_y_freq
        if self.entropy(total) == 0: # see if it is pure
            return -1
        else:
            infor_gain = self.entropy(total) - self.entropy_impurity(left_y_freq, right
```

```
_y_freq)
        return infor_gain

#     @staticmethod
#     def gini(y):
#         p = y / (np.sum(y)+1e-20)
#         gini = 1-np.sum(p**2)
#         return gini

#     @staticmethod
#     def gini_impurity(left_label_freq, right_label_freq): # useless
#         Sl = np.sum(left_label_freq)
#         Sr = np.sum(right_label_freq)
#         return (Sl * gini(left_label_freq) + Sr * gini(right_label_freq)) / (Sl+Sr)

#     @staticmethod
#     def gini_purification(X, y, thresh):
#         """
#         TODO: implement a method that calculates reduction in impurity gain given a v
ector of features
#         and a split threshold
#         """
#         return 0

    def split(self, S, depth, random_f = -1, verbose = False): # recursively
        """
        TODO: implement a method that return a split of the dataset given an index of t
he feature and
        a threshold for it
        """
#         print((depth-1) * '    '+'Depth: '+ str(depth))
        if depth >= self.max_depth:
#             print('label: '+ str(self.max_count(self.labels[S])))
            node = self.Node(left=None, right=None, split_rule=None, is_leaf=1, label=s
elf.max_count(self.labels[S]))
            if verbose == True:
                print(repr(node))
            return node
        else:
            max_feature, max_thresh = self.segmenter(self.data[S, :], self.labels[S],ra
ndom_f = random_f)

#             print((depth-1) * '    '+'depth:' + str(depth)  + ', feature index: ' + s
tr(max_feature) +', threshold: ' + str(max_thresh))
            Sl = [i for i in S if self.data[i, max_feature] <= max_thresh]
            Sr = [i for i in S if self.data[i, max_feature] > max_thresh]
#             print((depth-1) * '    '+"left group: " + str(len(Sl)) + ', right group:
 ' + str(len(Sr)))
            if len(Sl) <= 5 or len(Sr) <= 5:
#                 print('label: '+ str(self.max_count(self.labels[S])))
                node = self.Node(left=None, right=None, split_rule=None, is_leaf=1, lab
el=self.max_count(self.labels[S]))
                if verbose == True:
                    print(repr(node))
                return node
            else:
                node = self.Node(left=self.split(Sl, depth+1, random_f), right=self.spl
it(Sr,depth+1,random_f), split_rule = (max_feature, max_thresh), is_leaf=0, label=None)
                if verbose == True:
                    print(repr(node))
                return node
```

```python
    def iter_thresh(self, X, y):
        """
        A method that return the max threshold and max information gain for one feature
        """
        row_f = sorted(set(X))
        col_l = set(y)
        freq_matrix = np.zeros([len(row_f), len(col_l)])
        for i, j in enumerate(row_f):
            for k, l in enumerate(col_l):
                freq_matrix[i, k] = len(y[np.where(y[np.where(X==j)]==l)])
        all_thresh = np.array(row_f[1:] + row_f[-1:]) / 2.
        left_freq = np.zeros([len(col_l)])
        right_freq = np.sum(freq_matrix, axis=0)
        left_freq_sum = 0
        max_thresh = all_thresh[0]

        max_gain = self.information_gain(left_freq, right_freq)
        for i, thresh in enumerate(all_thresh):
            left_freq += freq_matrix[i, :]
            right_freq -= freq_matrix[i, :]
            gain = self.information_gain(left_freq, right_freq)
            if gain > max_gain:
                max_gain = gain
                max_thresh = thresh
        return max_thresh, max_gain


    def segmenter(self, X, y, random_f = -1):
        """
        TODO: compute entropy gain for all single-dimension splits,
        return the feature and the threshold for the split that
        has maximum gain
        """
        x = X.shape[1]
        if random_f == -1:
            all_features = np.arange(x)
        else:
            all_features = np.random.choice(range(x), random_f)

        all_features = np.arange(x)
        max_gain = 1e-10
        max_thresh = 0
        max_feature = 0
        for i in all_features:
            thresh, gain = self.iter_thresh(X[:, i], y)
            if gain > max_gain:
                max_gain = gain
                max_thresh = thresh
                max_feature = i
        return max_feature, max_thresh


    def fit(self, X, y, random_f = -1, verbose = True):
        """
        TODO: fit the model to a training set. Think about what would be
        your stopping criteria
        """
        self.data = X
        self.labels = y
```

```
        S = np.array(range(len(y)))
        self.root = self.split(S, 1 , random_f = random_f, verbose = verbose)
        return self


    def predict(self, X, T = 0):
        """
        TODO: predict the labels for input data
        """
        if T == 1:  # T = 1, for random forest
            X = np.reshape(X, [1, len(X)])
            row_num = 1
        else:
            row_num = X.shape[0]
        labels = np.zeros(row_num)
        depth = 0
        for i in range(row_num):
            current_node = self.root
            while current_node.is_leaf == 0:
                feature = current_node.split_rule[0]
                thresh = current_node.split_rule[1]
                if X[i,:][feature] <= thresh:
                    current_node = current_node.left
                else:
                    current_node = current_node.right
            depth += 1
            labels[i] = current_node.label
        return labels

    def accuracy(self, X, y_val, T = 0):
        y_pred = self.predict(X, T = T)
        len_y = float(len(y_pred))
        return np.sum(y_pred == y_val) / len_y
```

1.2 Implement Random Forests

In [3]:

```
class RandomForest():

    def __init__(self, n_trees=20, n_sample=1000, random_f=-1, max_depth=200):
        """
        TODO: initialization of a random forest
        """
        self.n_trees = n_trees
        self.n_sample = n_sample
        self.random_f = random_f
        self.max_depth = max_depth
        self.trees = np.array([DecisionTree(max_depth)] * n_trees)

    def fit(self, X, y):
        """
        TODO: fit the model to a training set.
        """
        if self.random_f == -1:
            self.random_f = int(np.sqrt(X.shape[1]))
        results = np.zeros(self.n_trees, dtype=object)
        for i, dt in enumerate(self.trees):
            print('#%d. tree' % i)
            idx = np.random.choice(range(len(X)), self.n_sample)
            sub_X = X[idx, :]
            sub_y = y[idx]
            results[i] = dt.fit(sub_X, sub_y, random_f=self.random_f, verbose = False)
        self.trees = results

    def predict(self, X, T = 1):
        """
        TODO: predict the labels for input data
        """
        row_num = X.shape[0]
        labels = []
        for i in range(row_num):
            pred = np.zeros(self.n_trees)
            for j, dt in enumerate(self.trees):
                pred[j] = dt.predict(X[i, :], T = T)
            labels.append(dt.max_count(pred))
        return labels

    def accuracy(self, X, y_val, T = 1):
        y_pred = self.predict(X,T = T)
        N = float(len(y_pred))
        return np.sum(y_pred == y_val) / N
```

1.3 Describe implementation details

1. I one-hot encode the categorical features and use median for numerical data and mode for categorical data to impute missing values. For age column, I group by the number of family member and impute the mean age.

2. I have three stop criterion: when the depth of the tree is deeper than the max depth, when parent group is pure (or the entropy is 0) and when either group has less than 5 points.

3. I implement the random forest by bagging. Each tree of this model randomly select samples and features of the data. The final predicted label of a point is the mode of all the predicted labels of the trees.

4. In order to speed up training, I use 'pool' or 'multiprocessing'.

5. I print out the structure of decision tree.

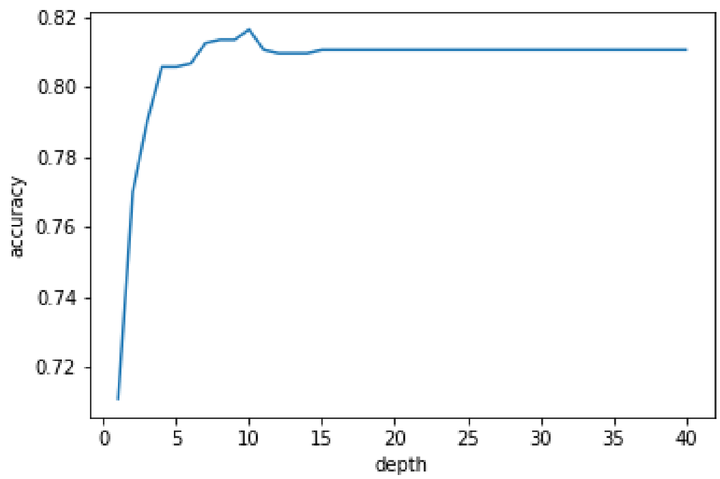
1.4 Performance Evaluation

1. Decision Tree:
The training accuracy of spam:0.8286197727822093
The validation accuracy of spam:0.8144927536231884
The training accuracy of titanic:0.77625
The validation accuracy of titanic:0.84

2. Random Forest:
The training accuracy of spam:0.8300700991056321
The validation accuracy of spam:0.8241545893719807
The training accuracy of titanic:0.7825
The validation accuracy of titanic:0.84

1.5 Writeup Requirements for the Spam Dataset

3.



From 1 to around 15 depth, the validation accuracy is increasing fast, after 15 depth, the accuracy is stable and remains around 0.81.

In [1]:

```
from collections import Counter
import numpy as np
from numpy import genfromtxt
import scipy.io
from scipy import stats
import random
import pandas as pd
from statistics import mode
import pydot
```

In [2]:

```
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
class_names = ["Ham", "Spam"]
```

In [21]:

```
class DecisionTree:

    class Node:
        def __init__(self, split_rule, left, right, label, is_leaf):
            self.split_rule = split_rule
            self.left = left
            self.right = right
            self.label = label
            self.is_leaf = is_leaf # 1 = stop

        def __repr__(self):
            """
            TODO: one way to visualize the decision tree is to write out a __repr__ met
hod
            that returns the string representation of a tree. Think about how to visual
ize
            a tree structure. You might have seen this before in CS61A.
            """
            def viz(Node, prefix, symbol):
                if not Node:
                    return prefix + '[]'
                if Node.is_leaf == 1:
                    return(prefix + '(Therefore the email was: '+ class_names[Node.labe
l] + ')')
                else:
                    ret = (prefix + '[Feature: '+ features[Node.split_rule[0]] +
                           ', Threshold: '+symbol+ str(Node.split_rule[1]) + ']')
                    ret += '\n' + viz(Node.left, prefix + '\t','<=') + '\n' + viz(Node.
right, prefix + '\t','>')
                    return ret

            return viz(self, "",'<=')


    def __init__(self, max_depth = 200):
        self.max_depth = max_depth

    def max_count(self, array):
        return stats.mode(array, nan_policy='omit')[0][0]


    def entropy(self,y):
        p = y / (np.sum(y)+1e-10)
        return -p.dot(np.log2(p+1e-10))


    def entropy_impurity(self,left_y_freq, right_y_freq):
        Sl = np.sum(left_y_freq)
        Sr = np.sum(right_y_freq)
        return (Sl * self.entropy(left_y_freq) + Sr * self.entropy(right_y_freq)) / (Sl
+Sr)


    def information_gain(self,left_y_freq, right_y_freq):
        total = left_y_freq + right_y_freq
        if self.entropy(total) == 0: # see if it is pure
            return -1
        else:
            infor_gain = self.entropy(total) - self.entropy_impurity(left_y_freq, right
```

```
_y_freq)
        return infor_gain

#     @staticmethod
#     def gini(y):
#         p = y / (np.sum(y)+1e-20)
#         gini = 1-np.sum(p**2)
#         return gini

#     @staticmethod
#     def gini_impurity(left_label_freq, right_label_freq): # useless
#         Sl = np.sum(left_label_freq)
#         Sr = np.sum(right_label_freq)
#         return (Sl * gini(left_label_freq) + Sr * gini(right_label_freq)) / (Sl+Sr)

#     @staticmethod
#     def gini_purification(X, y, thresh):
#         """
#         TODO: implement a method that calculates reduction in impurity gain given a v
ector of features
#         and a split threshold
#         """
#         return 0

    def split(self, S, depth, random_f = -1, verbose = False): # recursively
        """
        TODO: implement a method that return a split of the dataset given an index of t
he feature and
        a threshold for it
        """
#         print((depth-1) * '     '+'Depth: '+ str(depth))
        if depth >= self.max_depth:
#             print('label: '+ str(self.max_count(self.labels[S])))
            node = self.Node(left=None, right=None, split_rule=None, is_leaf=1, label=s
elf.max_count(self.labels[S]))
            if verbose == True:
                print(repr(node))
            return node
        else:
            max_feature, max_thresh = self.segmenter(self.data[S, :], self.labels[S],ra
ndom_f = random_f)

#             print((depth-1) * '     '+'depth:' + str(depth)  + ', feature index: ' + s
tr(max_feature) +', threshold: ' + str(max_thresh))
            Sl = [i for i in S if self.data[i, max_feature] <= max_thresh]
            Sr = [i for i in S if self.data[i, max_feature] > max_thresh]
#             print((depth-1) * '     '+"left group: " + str(len(Sl)) + ', right group:
 ' + str(len(Sr)))
            if len(Sl) <= 5 or len(Sr) <= 5:
#                 print('label: '+ str(self.max_count(self.labels[S])))
                node = self.Node(left=None, right=None, split_rule=None, is_leaf=1, lab
el=self.max_count(self.labels[S]))
                if verbose == True:
                    print(repr(node))
                return node
            else:
                node = self.Node(left=self.split(Sl, depth+1, random_f), right=self.spl
it(Sr,depth+1,random_f), split_rule = (max_feature, max_thresh), is_leaf=0, label=None)
                if verbose == True:
                    print(repr(node))
                return node
```

```
    def iter_thresh(self, X, y):
        """
        A method that return the max threshold and max information gain for one feature
        """
        row_f = sorted(set(X))
        col_l = set(y)
        freq_matrix = np.zeros([len(row_f), len(col_l)])
        for i, j in enumerate(row_f):
            for k, l in enumerate(col_l):
                freq_matrix[i, k] = len(y[np.where(y[np.where(X==j)]==l)])
        all_thresh = np.array(row_f[1:] + row_f[-1:]) / 2.
        left_freq = np.zeros([len(col_l)])
        right_freq = np.sum(freq_matrix, axis=0)
        left_freq_sum = 0
        max_thresh = all_thresh[0]

        max_gain = self.information_gain(left_freq, right_freq)
        for i, thresh in enumerate(all_thresh):
            left_freq += freq_matrix[i, :]
            right_freq -= freq_matrix[i, :]
            gain = self.information_gain(left_freq, right_freq)
            if gain > max_gain:
                max_gain = gain
                max_thresh = thresh
        return max_thresh, max_gain


    def segmenter(self, X, y, random_f = -1):
        """
        TODO: compute entropy gain for all single-dimension splits,
        return the feature and the threshold for the split that
        has maximum gain
        """
        x = X.shape[1]
        if random_f == -1:
            all_features = np.arange(x)
        else:
            all_features = np.random.choice(range(x), random_f)

        all_features = np.arange(x)
        max_gain = 1e-10
        max_thresh = 0
        max_feature = 0
        for i in all_features:
            thresh, gain = self.iter_thresh(X[:, i], y)
            if gain > max_gain:
                max_gain = gain
                max_thresh = thresh
                max_feature = i
        return max_feature, max_thresh


    def fit(self, X, y, random_f = -1, verbose = True):
        """
        TODO: fit the model to a training set. Think about what would be
        your stopping criteria
        """
        self.data = X
        self.labels = y
```

```
        S = np.array(range(len(y)))
        self.root = self.split(S, 1 , random_f = random_f, verbose = verbose)
        return self


    def predict(self, X, T = 0):
        """
        TODO: predict the labels for input data
        """
        if T == 1:  # T = 1, for random forest
            X = np.reshape(X, [1, len(X)])
            row_num = 1
        else:
            row_num = X.shape[0]
        labels = np.zeros(row_num)
        depth = 0
        for i in range(row_num):
            current_node = self.root
            while current_node.is_leaf == 0:
                feature = current_node.split_rule[0]
                thresh = current_node.split_rule[1]
                if X[i,:][feature] <= thresh:
                    current_node = current_node.left
                else:
                    current_node = current_node.right
            depth += 1
            labels[i] = current_node.label
        return labels

    def accuracy(self, X, y_val, T = 0):
        y_pred = self.predict(X, T = T)
        len_y = float(len(y_pred))
        return np.sum(y_pred == y_val) / len_y
```

In [22]:

```
class RandomForest():

    def __init__(self, n_trees=20, n_sample=1000, random_f=-1, max_depth=200):
        """
        TODO: initialization of a random forest
        """
        self.n_trees = n_trees
        self.n_sample = n_sample
        self.random_f = random_f
        self.max_depth = max_depth
        self.trees = np.array([DecisionTree(max_depth)] * n_trees)

    def fit(self, X, y):
        """
        TODO: fit the model to a training set.
        """
        if self.random_f == -1:
            self.random_f = int(np.sqrt(X.shape[1]))
        results = np.zeros(self.n_trees, dtype=object)
        for i, dt in enumerate(self.trees):
            print('#%d. tree' % i)
            idx = np.random.choice(range(len(X)), self.n_sample)
            sub_X = X[idx, :]
            sub_y = y[idx]
            results[i] = dt.fit(sub_X, sub_y, random_f=self.random_f, verbose = False)
        self.trees = results

    def predict(self, X, T = 1):
        """
        TODO: predict the labels for input data
        """
        row_num = X.shape[0]
        labels = []
        for i in range(row_num):
            pred = np.zeros(self.n_trees)
            for j, dt in enumerate(self.trees):
                pred[j] = dt.predict(X[i, :], T = T)
            labels.append(dt.max_count(pred))
        return labels

    def accuracy(self, X, y_val, T = 1):
        y_pred = self.predict(X,T = T)
        N = float(len(y_pred))
        return np.sum(y_pred == y_val) / N
```

In [5]:

```
def normalization(X):
    Xn = np.zeros(X.shape)
    for i in range(X.shape[0]):
        x = X[i, :]
        Xn[i, :] = (x- np.min(x)/(np.max(x)-np.min(x)))
    return Xn
```

In [25]:

```
# load spam data
spam = scipy.io.loadmat('datasets/spam-dataset/spam_data.mat')
train_X = spam['training_data']
train_y = spam['training_labels'].ravel()
test_X = spam['test_data']
```

In [26]:

```
# split data
idx = np.random.choice(range(len(train_y)), int(len(train_y)), replace=False)
train_size = int(len(train_X)*0.8)
X_train = train_X[idx,:][:train_size,:]
y_train = train_y[idx][:train_size]
X_val = train_X[idx,:][train_size:,:]
y_val = train_y[idx][train_size:]
```

In [29]:

```
# try training
dt = DecisionTree(16)
dt.fit(X_train, y_train, verbose = True)
print(repr(dt))
train_acc = dt.accuracy(X_train, y_train)
print('Training accuracy: ', train_acc)
val_acc = dt.accuracy(X_val, y_val)
print('Validation accuracy: ', val_acc)
```

```
[Feature: exclamation, Threshold: <=0.5]
        [Feature: meter, Threshold: <=0.5]
                [Feature: parenthesis, Threshold: <=0.5]
                        [Feature: ampersand, Threshold: <=0.5]
                                [Feature: volumes, Threshold: <=0.5]
                                        [Feature: semicolon, Threshold: <=
0.5]
                                                [Feature: pain, Threshold:
<=0.5]
                                                        [Feature: square_b
racket, Threshold: <=0.5]
                                                                [Feature:
prescription, Threshold: <=0.5]
                                                                        [F
eature: energy, Threshold: <=0.5]

[Feature: drug, Threshold: <=0.5]

(Therefore the email was: Ham)

(Therefore the email was: Spam)

[Feature: energy, Threshold: >1.0]

[Feature: dollar, Threshold: <=0.5]

(Therefore the email was: Ham)

(Therefore the email was: Ham)

(Therefore the email was: Ham)
                                                                        (T
herefore the email was: Spam)
                                                                (Therefore
the email was: Spam)
                                                        (Therefore the ema
il was: Spam)
                                                [Feature: dollar, Threshol
d: >0.5]
                                                        [Feature: square_b
racket, Threshold: <=0.5]
                                                                (Therefore
the email was: Ham)
                                                                (Therefore
the email was: Ham)
                                                        (Therefore the ema
il was: Spam)
                                        (Therefore the email was: Ham)
                                (Therefore the email was: Ham)
                        [Feature: dollar, Threshold: >0.5]
                                [Feature: featured, Threshold: <=0.5]
                                        (Therefore the email was: Ham)
                                        (Therefore the email was: Spam)
                                [Feature: private, Threshold: >0.5]
                                        (Therefore the email was: Ham)
                                        (Therefore the email was: Spam)
                (Therefore the email was: Ham)
        [Feature: meter, Threshold: >0.5]
                [Feature: ampersand, Threshold: <=0.5]
                        [Feature: money, Threshold: <=0.5]
                                [Feature: dollar, Threshold: <=1.0]
```

```
[Feature: prescription, Threshold:
<=0.5]
[Feature: parenthesis, Thr
eshold: <=0.5]
[Feature: pain, Th
reshold: <=0.5]
[Feature:
message, Threshold: <=0.5]
[F
eature: other, Threshold: <=0.5]
[Feature: semicolon, Threshold: <=0.5]
[Feature: sharp, Threshold: <=1.0]
[Feature: square_bracket, Threshold: <=0.5]
[Feature: dollar, Threshold: <=0.5]
(Therefore the email was: Spam)
(Therefore the email was: Spam)
(Therefore the email was: Spam)
(Therefore the email was: Ham)
[Feature: exclamation, Threshold: >1.0]
(Therefore the email was: Spam)
(Therefore the email was: Spam)
(Therefore the email was: Ham)
[F
eature: semicolon, Threshold: >1.0]
(Therefore the email was: Spam)
(Therefore the email was: Ham)
(Therefore
the email was: Spam)
[Feature: spam, Th
reshold: >0.5]
[Feature:
message, Threshold: <=0.5]
[F
eature: drug, Threshold: <=0.5]
[Feature: featured, Threshold: <=0.5]
[Feature: volumes, Threshold: <=0.5]
[Feature: semicolon, Threshold: <=0.5]
(Therefore the email was: Ham)
(Therefore the email was: Spam)
(Therefore the email was: Ham)
```

```
(Therefore the email was: Spam)

(Therefore the email was: Spam)
                                                                             [F
eature: semicolon, Threshold: >0.5]

[Feature: parenthesis, Threshold: <=2.0]

(Therefore the email was: Ham)

(Therefore the email was: Ham)

(Therefore the email was: Ham)
                                                                     (Therefore
the email was: Spam)
                                                     (Therefore the email was:
Spam)
                                             [Feature: message, Threshold: >2.
5]
                                                     (Therefore the email was:
Spam)
                                                     (Therefore the email was:
Spam)
                                 (Therefore the email was: Spam)
                         [Feature: other, Threshold: >0.5]
                                 (Therefore the email was: Ham)
                                 (Therefore the email was: Ham)
                 (Therefore the email was: Ham)
<__main__.DecisionTree object at 0x000001EF5B84F400>
Training accuracy:  0.8286197727822093
Validation accuracy:  0.8144927536231884
```

In [32]:

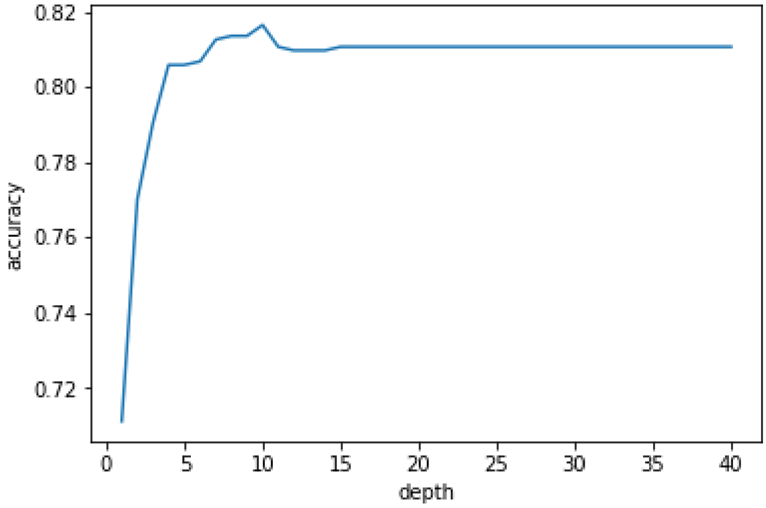
```
accuracy_list = []
for i in range(40):
    print(str(i+1))
    dt = DecisionTree(i+1)
    dt.fit(X_train, y_train, verbose = False)
    acc = dt.accuracy(X_val, y_val)
    accuracy_list.append(acc)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
```

In [12]:

```
# accuracy_list
mylist = np.linspace(1,40,40)
import matplotlib.pyplot as plt
fig = plt.figure()
plt.plot(mylist,accuracy_list)
plt.xlabel('depth')
plt.ylabel('accuracy')
fig.savefig('1-40depth-accuracy-dt.png')

# when the depth is equal or larger than 15, the validation accuracy is stable and the
 largest
```



In [34]:

```
rf = RandomForest(n_trees=20, n_sample=len(y_train), random_f=-1, max_depth=20)
rf.fit(X_train, y_train) # use all data
train_acc = rf.accuracy(X_train,y_train)
print('Training accuracy: ', train_acc)
val_acc = rf.accuracy(X_val,y_val)
print('Validation accuracy: ', val_acc)
```

```
#0. tree
#1. tree
#2. tree
#3. tree
#4. tree
#5. tree
#6. tree
#7. tree
#8. tree
#9. tree
#10. tree
#11. tree
#12. tree
#13. tree
#14. tree
#15. tree
#16. tree
#17. tree
#18. tree
#19. tree
Training accuracy:  0.8300700991056321
Validation accuracy:  0.8241545893719807
```

In [93]:

```
from collections import Counter
import numpy as np
from numpy import genfromtxt
import scipy.io
from scipy import stats
import random
import pandas as pd
from statistics import mode
import time
import multiprocessing as mp
import pydot
```

In [94]:

```
def normalization(X):
    Xn = np.zeros(X.shape)
    for i in range(X.shape[0]):
        x = X[i, :]
        Xn[i, :] = (x- np.min(x)/(np.max(x)-np.min(x)))
    return Xn
```

In [95]:

```
class_names = ["Ham", "Spam"]
```

In [96]:

```
class DecisionTree:

    class Node:
        def __init__(self, split_rule, left, right, label, is_leaf):
            self.split_rule = split_rule
            self.left = left
            self.right = right
            self.label = label
            self.is_leaf = is_leaf # 1 = stop

        def __repr__(self):
            """
            TODO: one way to visualize the decision tree is to write out a __repr__ met
hod
            that returns the string representation of a tree. Think about how to visual
ize
            a tree structure. You might have seen this before in CS61A.
            """
            def viz(Node, prefix, symbol):
                if not Node:
                    return prefix + '[]'
                if Node.is_leaf == 1:
                    return(prefix + '(Therefore the email was: '+ class_names[Node.labe
l] + ')')
                else:
                    ret = (prefix + '[Feature: '+ str(Node.split_rule[0]) +
                           ', Threshold: '+symbol+ str(Node.split_rule[1]) + ']')
                    ret += '\n' + viz(Node.left, prefix + '\t','<=') + '\n' + viz(Node.
right, prefix + '\t','>')
                    return ret

            return viz(self, "",'<=')


    def __init__(self, max_depth = 200):
        self.max_depth = max_depth

    def max_count(self, array):
        return stats.mode(array, nan_policy='omit')[0][0]

    def entropy(self,y):
        p = y / (np.sum(y)+1e-10)
        return -p.dot(np.log2(p+1e-10))

    def entropy_impurity(self,left_y_freq, right_y_freq):
        Sl = np.sum(left_y_freq)
        Sr = np.sum(right_y_freq)
        return (Sl * self.entropy(left_y_freq) + Sr * self.entropy(right_y_freq)) / (Sl
+Sr)

    def information_gain(self,left_y_freq, right_y_freq):
        total = left_y_freq + right_y_freq
        if self.entropy(total) == 0: # see if it is pure
            return -1
        else:
            infor_gain = self.entropy(total) - self.entropy_impurity(left_y_freq, right
_y_freq)
        return infor_gain
```

```
#     def gini(y):
#         p = y / (np.sum(y)+1e-20)
#         gini = 1-np.sum(p**2)
#         return gini

#     def gini_impurity(left_label_freq, right_label_freq): # useless
#         Sl = np.sum(left_label_freq)
#         Sr = np.sum(right_label_freq)
#         return (Sl * gini(left_label_freq) + Sr * gini(right_label_freq)) / (Sl+Sr)

#     def gini_purification(X, y, thresh):
#         """
#         TODO: implement a method that calculates reduction in impurity gain given a v
ector of features
#         and a split threshold
#         """
#         return 0

    def split(self, S, depth, random_f = -1,verbose = False): # recursively
        """
        TODO: implement a method that return a split of the dataset given an index of t
he feature and
        a threshold for it
        """
#         print((depth-1) * '    '+'Depth: '+ str(depth))
        if depth >= self.max_depth:
#             print('label: '+ str(self.max_count(self.labels[S])))
            node = self.Node(left=None, right=None, split_rule=None, is_leaf=1, label=s
elf.max_count(self.labels[S]))
            if verbose == True:
                print(repr(node))
            return node
        else:
            max_feature, max_thresh = self.segmenter(self.data[S, :], self.labels[S],ra
ndom_f = random_f)

#             print((depth-1) * '    '+'depth:' + str(depth)  + ', feature index: ' + s
tr(max_feature) +', threshold: ' + str(max_thresh))
            Sl = [i for i in S if self.data[i, max_feature] <= max_thresh]
            Sr = [i for i in S if self.data[i, max_feature] > max_thresh]
#             print((depth-1) * '    '+"left group: " + str(len(Sl)) + ', right group:
 ' + str(len(Sr)))
            if len(Sl) <= 5 or len(Sr) <= 5:
#                 print('label: '+ str(self.max_count(self.labels[S])))
                node = self.Node(left=None, right=None, split_rule=None, is_leaf=1, lab
el=self.max_count(self.labels[S]))
                if verbose == True:
                    print(repr(node))
                return node
            else:
                node = self.Node(left=self.split(Sl, depth+1, random_f), right=self.spl
it(Sr,depth+1,random_f), split_rule = (max_feature, max_thresh), is_leaf=0, label=None)
                if verbose == True:
                    print(repr(node))
                return node


    def iter_thresh(self, X, y):
        row_f = sorted(set(X))
        col_l = set(y)
        freq_matrix = np.zeros([len(row_f), len(col_l)])
```

```
        for i, j in enumerate(row_f):
            for k, l in enumerate(col_l):
                freq_matrix[i, k] = len(y[np.where(y[np.where(X==j)]==l)])
        all_thresh = np.array(row_f[1:] + row_f[-1:]) / 2.
        left_freq = np.zeros([len(col_l)])
        right_freq = np.sum(freq_matrix, axis=0)
        left_freq_sum = 0
        max_thresh = all_thresh[0]

        max_gain = self.information_gain(left_freq, right_freq)
        for i, thresh in enumerate(all_thresh):
            left_freq += freq_matrix[i, :]
            right_freq -= freq_matrix[i, :]
            gain = self.information_gain(left_freq, right_freq)
            if gain > max_gain:
                max_gain = gain
                max_thresh = thresh
        return max_thresh, max_gain


    def segmenter(self, X, y, random_f = -1):
        """
        TODO: compute entropy gain for all single-dimension splits,
        return the feature and the threshold for the split that
        has maximum gain
        """
        x = X.shape[1]
        if random_f == -1:
            all_features = np.arange(x)
        else:
            all_features = np.random.choice(range(x), random_f)

        all_features = np.arange(x)
        max_gain = 1e-10
        max_thresh = 0
        max_feature = 0
        for i in all_features:
            thresh, gain = self.iter_thresh(X[:, i], y)
            if gain > max_gain:
                max_gain = gain
                max_thresh = thresh
                max_feature = i
        return max_feature, max_thresh


    def fit(self, X, y, random_f = -1, verbose = False):
        """
        TODO: fit the model to a training set. Think about what would be
        your stopping criteria
        """
        self.data = X
        self.labels = y
        S = np.array(range(len(y)))
        self.root = self.split(S, 1 , random_f = random_f, verbose = verbose)
        return self


    def predict(self, X, T = 0):
        """
        TODO: predict the labels for input data
        """
```

```
        if T == 1: # T = 1, for random forest
            X = np.reshape(X, [1, len(X)])
            row_num = 1
        else:
            row_num = X.shape[0]
        labels = np.zeros(row_num)
        depth = 0
        for i in range(row_num):
            current_node = self.root
            while current_node.is_leaf == 0:
                feature = current_node.split_rule[0]
                thresh = current_node.split_rule[1]
                if X[i,:][feature] <= thresh:
                    current_node = current_node.left
                else:
                    current_node = current_node.right
            depth += 1
            labels[i] = current_node.label
        return labels

    def accuracy(self, X, y_val, T = 0):
        y_pred = self.predict(X, T = T)
        len_y = float(len(y_pred))
        return np.sum(y_pred == y_val) / len_y
```

In [97]:

```
class RandomForest():

    def __init__(self, n_trees=20, n_sample=1000, random_f=-1, max_depth=200):
        """
        TODO: initialization of a random forest
        """
        self.n_trees = n_trees
        self.n_sample = n_sample
        self.random_f = random_f
        self.max_depth = max_depth
        self.trees = np.array([DecisionTree(max_depth)] * n_trees)

    def fit(self, X, y, verbose = False):
        """
        TODO: fit the model to a training set.
        """
        if self.random_f == -1:
            self.random_f = int(np.log2(X.shape[1]))
        results = np.zeros(self.n_trees, dtype=object)
        for i, dt in enumerate(self.trees):
            print('#%d. tree' % (i+1))
            idx = np.random.choice(range(len(X)), self.n_sample)
            sub_X = X[idx, :]
            sub_y = y[idx]
            results[i] = dt.fit(sub_X, sub_y, random_f=self.random_f, verbose = verbose
)
        self.trees = results

    def predict(self, X, T = 1):
        """
        TODO: predict the labels for input data
        """
        row_num = X.shape[0]
        labels = []
        for i in range(row_num):
            pred = np.zeros(self.n_trees)
            for j, dt in enumerate(self.trees):
                pred[j] = dt.predict(X[i, :], T = T)
            labels.append(dt.max_count(pred))
        return labels

    def accuracy(self, X, y_val, T = 1):
        y_pred = self.predict(X,T = T)
        N = float(len(y_pred))
        return np.sum(y_pred == y_val) / N
```

In [98]:

```
# Load spam data
spam = scipy.io.loadmat('datasets/spam-dataset/spam_data.mat')
train_X = spam['training_data']
train_y = spam['training_labels'].ravel()
test_X = spam['test_data']
```

In [99]:

```
# split data
idx = np.random.choice(range(len(train_y)), int(len(train_y)), replace=False)
train_size = int(len(train_X)*0.8)
X_train = train_X[idx,:][:train_size,:]
y_train = train_y[idx][:train_size]
X_val = train_X[idx,:][train_size:,:]
y_val = train_y[idx][train_size:]
```

In [92]:

```
# try training
dt = DecisionTree(15)
dt.fit(X_train, y_train, verbose = True)
train_acc = dt.accuracy(X_train, y_train)
print('Training accuracy: ', train_acc)
val_acc = dt.accuracy(X_val, y_val)
print('Validation accuracy: ', val_acc)
```

```
[Feature: 28, Threshold: <=0.5]
        [Feature: 19, Threshold: <=0.5]
                [Feature: 29, Threshold: <=0.5]
                        [Feature: 16, Threshold: <=0.5]
                                [Feature: 0, Threshold: <=0.5]
                                        [Feature: 31, Threshold: <=0.5]
                                                [Feature: 25, Threshold: <
=0.5]
                                                        [Feature: 30, Thre
shold: <=0.5]
                                                                [Feature: 
6, Threshold: <=0.5]
                                                                        [F
eature: 13, Threshold: <=0.5]

[Feature: 20, Threshold: <=0.5]

(Therefore the email was: Ham)

(Therefore the email was: Ham)

[Feature: 13, Threshold: >1.0]

[Feature: 26, Threshold: <=0.5]

(Therefore the email was: Ham)

(Therefore the email was: Ham)

(Therefore the email was: Ham)
                                                                        (T
herefore the email was: Spam)
                                                                (Therefore
the email was: Spam)
                                                        [Feature: 26, Thre
shold: >0.5]
                                                                [Feature: 
25, Threshold: <=3.5]
                                                                        [F
eature: 24, Threshold: <=0.5]

[Feature: 15, Threshold: <=0.5]

(Therefore the email was: Ham)

(Therefore the email was: Ham)

(Therefore the email was: Ham)
                                                                        [F
eature: 15, Threshold: >0.5]

(Therefore the email was: Ham)

(Therefore the email was: Ham)
                                                                (Therefore
the email was: Spam)
                                                (Therefore the email was: 
Ham)
                                        (Therefore the email was: Spam)
                                (Therefore the email was: Ham)
                        [Feature: 26, Threshold: >0.5]
```

```
                                        (Therefore the email was: Ham)
                                        [Feature: 1, Threshold: >0.5]
                                                [Feature: 30, Threshold: <=2.0]
                                                        (Therefore the email was: 
Ham)
                                                        (Therefore the email was: 
Spam)
                                                (Therefore the email was: Spam)
                        (Therefore the email was: Ham)
                [Feature: 19, Threshold: >0.5]
                        [Feature: 31, Threshold: <=0.5]
                                [Feature: 3, Threshold: <=0.5]
                                        [Feature: 26, Threshold: <=1.0]
                                                [Feature: 6, Threshold: <=0.5]
                                                        [Feature: 16, Threshold: <
=0.5]
                                                                [Feature: 15, Thre
shold: <=0.5]
                                                                        [Feature: 
25, Threshold: <=0.5]
                                                                                [F
eature: 0, Threshold: <=0.5]

[Feature: 29, Threshold: <=0.5]

[Feature: 27, Threshold: <=1.0]

(Therefore the email was: Spam)

(Therefore the email was: Ham)

(Therefore the email was: Ham)

(Therefore the email was: Spam)
                                                                                [F
eature: 27, Threshold: >0.5]

[Feature: 9, Threshold: <=0.5]

(Therefore the email was: Spam)

(Therefore the email was: Spam)

(Therefore the email was: Spam)
                                                                        [Feature: 
25, Threshold: >5.0]
                                                                                (T
herefore the email was: Ham)
                                                                                (T
herefore the email was: Ham)
                                                                (Therefore the ema
il was: Ham)
                                                        (Therefore the email was: 
Spam)
                                                [Feature: 15, Threshold: >2.5]
                                                        (Therefore the email was: 
Spam)
                                                        [Feature: 5, Threshold: >
0.5]
                                                                (Therefore the ema
il was: Ham)
```

```
                                                      (Therefore the ema
il was: Spam)
                            [Feature: 28, Threshold: >2.0]
                                    (Therefore the email was: Spam)
                                    [Feature: 29, Threshold: >1.0]
                                            (Therefore the email was:
Spam)
                                            (Therefore the email was:
Spam)
                    [Feature: 3, Threshold: >0.5]
                            [Feature: 15, Threshold: <=0.5]
                                    [Feature: 12, Threshold: <=0.5]
                                            (Therefore the email was:
Ham)
                                            (Therefore the email was:
Ham)
                                    (Therefore the email was: Ham)
                            (Therefore the email was: Spam)
                (Therefore the email was: Ham)
Training accuracy:  0.8271694464587865
Validation accuracy:  0.8154589371980676
```

In [9]:

```
y_pred = dt.predict(test_X)
y_pred = np.array(y_pred)
y_pred
```

Out[9]:

```
array([0., 0., 0., ..., 0., 0., 0.])
```

In [10]:

```
# save
def results_to_csv(y_test):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1  # Ensures that the index starts at 1.
    df.to_csv('submission_spam_low2.csv', index_label='Id')

results_to_csv(y_pred)
```

In [100]:

```
# bag words model
from sklearn.feature_extraction.text import CountVectorizer
import glob

BASE_DIR = 'datasets/spam-dataset/'
SPAM_DIR = 'spam/'
HAM_DIR = 'ham/'
TEST_DIR = 'test/'

NUM_TRAINING_EXAMPLES = 5172
NUM_TEST_EXAMPLES = 5857

spam_filenames = glob.glob(BASE_DIR + SPAM_DIR + '*.txt')
ham_filenames = glob.glob(BASE_DIR + HAM_DIR + '*.txt')
test_filenames = [BASE_DIR + TEST_DIR + str(x) + '.txt' for x in range(NUM_TEST_EXAMPLE
S)]

train_text = []
for file in spam_filenames+ham_filenames:
    with open(file, "r", encoding='utf-8', errors='ignore') as f:
        train_text.append(f.read())

test_text = []
for file in test_filenames:
    with open(file, "r", encoding='utf-8', errors='ignore') as f:
        test_text.append(f.read())

vectorizer = CountVectorizer()
train_X = normalization(vectorizer.fit_transform(train_text).toarray())
test_X = normalization(vectorizer.transform(test_text).toarray())
train_y = np.concatenate((np.ones(len(spam_filenames)), np.zeros(len(ham_filenames))))
train_y = train_y.astype('int64')
```

In [11]:

```
# # feature selection
# std = np.std(train_X, axis=0)
# idx = std.argsort()[-5000:]
# train_X_selected = train_X[:, idx]
# test_X_selected = test_X[:, idx]
```

In [101]:

```
from sklearn.feature_selection import SelectKBest, chi2
best = SelectKBest(score_func=chi2, k=5000)
fit = best.fit(train_X,train_y)
```

In [102]:

```
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(pd.DataFrame(train_X).columns)

featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']
topindex = featureScores['Score'].sort_values(ascending = False)[:5000].index
topindex = topindex.tolist()
```

In [103]:

```
train_X_selected = train_X[:,topindex]
test_X_selected = test_X[:,topindex]
```

In [ ]:

```

```

In [104]:

```
# data split
idx = np.random.choice(range(len(train_y)),len(train_y),replace = False)
split = int(len(train_y)*0.8)
X_train,X_val = train_X_selected[idx][:split,],train_X_selected[idx][split:,]
y_train,y_val = train_y[idx][:split],train_y[idx][split:]
```

In [105]:

```
# try training
dt = DecisionTree(15)
dt.fit(X_train, y_train, verbose = True)
train_acc = dt.accuracy(X_train, y_train)
print('Training accuracy: ', train_acc)
val_acc = dt.accuracy(X_val, y_val)
print('Validation accuracy: ', val_acc)
```

```
[Feature: 2, Threshold: <=0.5]
        [Feature: 12, Threshold: <=0.5]
                [Feature: 19, Threshold: <=0.5]
                        [Feature: 4, Threshold: <=0.5]
                                [Feature: 31, Threshold: <=0.5]
                                        [Feature: 11, Threshold: <=0.5]
                                                [Feature: 58, Threshold: <
=0.5]
                                                        [Feature: 18, Thre
shold: <=0.5]
                                                                [Feature:
17, Threshold: <=0.5]
                                                                        [F
eature: 70, Threshold: <=0.5]

[Feature: 1664, Threshold: <=0.5]

[Feature: 93, Threshold: <=0.5]

[Feature: 22, Threshold: <=0.5]

[Feature: 9, Threshold: <=0.5]

(Therefore the email was: Spam)

(Therefore the email was: Ham)

[Feature: 94, Threshold: >0.5]

(Therefore the email was: Spam)

(Therefore the email was: Spam)

[Feature: 35, Threshold: >1.5]

[Feature: 2105, Threshold: <=0.5]

(Therefore the email was: Spam)

(Therefore the email was: Ham)

[Feature: 39, Threshold: >4.0]

(Therefore the email was: Spam)

(Therefore the email was: Spam)

(Therefore the email was: Ham)

(Therefore the email was: Ham)
                                                                        [F
eature: 316, Threshold: >2.5]

(Therefore the email was: Ham)

(Therefore the email was: Ham)
                                                                [Feature:
35, Threshold: >5.0]
                                                                        (T
herefore the email was: Ham)
                                                                        (T
```

```
herefore the email was: Spam)
                                                        [Feature: 7, Thres
hold: >7.0]
                                                                (Therefore
the email was: Ham)
                                                                (Therefore
the email was: Spam)
                                                (Therefore the email was:
Ham)
                                        [Feature: 48, Threshold: >0.5]
                                                [Feature: 41, Threshold: <
=0.5]
                                                        [Feature: 0, Thres
hold: <=0]
                                                                (Therefore
the email was: Ham)
                                                                (Therefore
the email was: Ham)
                                                        (Therefore the ema
il was: Ham)
                                                [Feature: 39, Threshold: >
3.5]
                                                        [Feature: 1662, Th
reshold: <=0.5]
                                                                [Feature:
30, Threshold: <=0.5]
                                                                        (T
herefore the email was: Ham)
                                                                        (T
herefore the email was: Spam)
                                                                (Therefore
the email was: Ham)
                                                        (Therefore the ema
il was: Spam)
                                [Feature: 1007, Threshold: >1.0]
                                        [Feature: 1121, Threshold: <=0.5]
                                                [Feature: 1527, Threshold:
<=1.0]
                                                        [Feature: 255, Thr
eshold: <=1.0]
                                                                (Therefore
the email was: Spam)
                                                                [Feature:
396, Threshold: >1.5]
                                                                        [F
eature: 30, Threshold: <=0.5]
(Therefore the email was: Spam)
(Therefore the email was: Spam)
                                                                        (T
herefore the email was: Ham)
                                                        (Therefore the ema
il was: Ham)
                                                (Therefore the email was:
Ham)
                                        (Therefore the email was: Ham)
                        [Feature: 0, Threshold: >0]
                                (Therefore the email was: Ham)
                                (Therefore the email was: Ham)
                [Feature: 0, Threshold: >0]
```

```
                        (Therefore the email was: Ham)
                        (Therefore the email was: Ham)
        [Feature: 0, Threshold: >0]
                (Therefore the email was: Ham)
                (Therefore the email was: Ham)
Training accuracy:  0.9395697365240513
Validation accuracy:  0.9352657004830918
```

In [ ]:

```
# train
rf = RandomForest(n_trees=20, n_sample=len(y_train), random_f=-1, max_depth=20)
rf.fit(X_train, y_train, verbose = False)
```

```
#1. tree
#2. tree
#3. tree
#4. tree
#5. tree
#6. tree
#7. tree
#8. tree
#9. tree
#10. tree
#11. tree
#12. tree
#13. tree
#14. tree
#15. tree
```

In [15]:

```
l = rf.accuracy(X_train, y_train)
print('Accuracy: ', l)
```

```
Accuracy:  0.9402948996857626
```

In [16]:

```
# validation accuracy
l = rf.accuracy(X_val, y_val)
print('Accuracy: ', l)
```

```
Accuracy:  0.9352657004830918
```

In [17]:

```
# for submision
y_pred = rf.predict(test_X_selected)
y_pred = np.array(y_pred)
y_pred
```

Out[17]:

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

In [21]:

```
# save
def results_to_csv(y_test):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1  # Ensures that the index starts at 1.
    df.to_csv('submission_spam.csv', index_label='Id')

results_to_csv(y_pred)
```

1.6 Writeup Requirements for the Titanic Dataset

In [79]:

```
import numpy as np
from numpy import genfromtxt
import scipy.io
from scipy import stats
import matplotlib.pyplot as plt
import random
import pandas as pd
import pydot
from scipy.stats import hmean
from scipy.spatial.distance import cdist
import numbers
from collections import Counter,defaultdict

features = ['pclass','age','sibsp','parch','fare','sex','embarked_C','embarked_Q','emba
rked_S']
class_names = ['died','survived']

class DecisionTree:

    class Node:
        def __init__(self, split_rule, left, right, label, is_leaf):
            self.split_rule = split_rule
            self.left = left
            self.right = right
            self.label = label
            self.is_leaf = is_leaf # 1 = stop

        def __repr__(self):
            """
            TODO: one way to visualize the decision tree is to write out a __repr__ met
hod
            that returns the string representation of a tree. Think about how to visual
ize
            a tree structure. You might have seen this before in CS61A.
            """
            def viz(Node, prefix, symbol):
                if not Node:
                    return prefix + '[]'
                if Node.is_leaf == 1:
                    return(prefix + '(Therefore the person was: '+ class_names[Node.lab
el] + ')')
                else:
                    ret = (prefix + '[Feature: '+ str(Node.split_rule[0]) +
                           ', Threshold: '+symbol+ str(Node.split_rule[1]) + ']')
                    ret += '\n' + viz(Node.left, prefix + '\t','<=') + '\n' + viz(Node.
right, prefix + '\t','>')
                    return ret

            return viz(self, "",'<=')


    def __init__(self, max_depth = 200):
        self.max_depth = max_depth

    def max_count(self, array):
        return stats.mode(array, nan_policy='omit')[0][0]

    def entropy(self,y):
        p = y / (np.sum(y)+1e-10)
```

```
        return -p.dot(np.log2(p+1e-10))

    def entropy_impurity(self,left_y_freq, right_y_freq):
        Sl = np.sum(left_y_freq)
        Sr = np.sum(right_y_freq)
        return (Sl * self.entropy(left_y_freq) + Sr * self.entropy(right_y_freq)) / (Sl
+Sr)

    def information_gain(self,left_y_freq, right_y_freq):
        total = left_y_freq + right_y_freq
        if self.entropy(total) == 0: # see if it is pure
            return -1
        else:
            infor_gain = self.entropy(total) - self.entropy_impurity(left_y_freq, right
_y_freq)
        return infor_gain

#     @staticmethod
#     def gini(y):
#         p = y / (np.sum(y)+1e-20)
#         gini = 1-np.sum(p**2)
#         return gini

#     @staticmethod
#     def gini_impurity(left_label_freq, right_label_freq): # useless
#         Sl = np.sum(left_label_freq)
#         Sr = np.sum(right_label_freq)
#         return (Sl * gini(left_label_freq) + Sr * gini(right_label_freq)) / (Sl+Sr)

#     @staticmethod
#     def gini_purification(X, y, thresh):
#         """
#         TODO: implement a method that calculates reduction in impurity gain given a v
ector of features
#         and a split threshold
#         """
#         return 0

    def split(self, S, depth, random_f = -1, verbose = False): # recursively
        """
        TODO: implement a method that return a split of the dataset given an index of t
he feature and
        a threshold for it
        """
#         print((depth-1) * '     '+'Depth: '+ str(depth))
        if depth >= self.max_depth:
#             print('label: '+ str(self.max_count(self.labels[S])))
            node = self.Node(left=None, right=None, split_rule=None, is_leaf=1, label=s
elf.max_count(self.labels[S]))
            if verbose == True:
                print(repr(node))
            return node
        else:
            max_feature, max_thresh = self.segmenter(self.data[S, :], self.labels[S],ra
ndom_f = random_f)

#             print((depth-1) * '     '+'depth:' + str(depth)  + ', feature index: ' + s
tr(max_feature) +', threshold: ' + str(max_thresh))
            Sl = [i for i in S if self.data[i, max_feature] <= max_thresh]
            Sr = [i for i in S if self.data[i, max_feature] > max_thresh]
#             print((depth-1) * '     '+"left group: " + str(len(Sl)) + ', right group:
```

```
 ' + str(len(Sr)))
            if len(Sl) <= 5 or len(Sr) <= 5:
#                 print('label: '+ str(self.max_count(self.labels[S])))
                node = self.Node(left=None, right=None, split_rule=None, is_leaf=1, lab
el=self.max_count(self.labels[S]))
                if verbose == True:
                    print(repr(node))
                return node
            else:
                node = self.Node(left=self.split(Sl, depth+1, random_f), right=self.spl
it(Sr,depth+1,random_f), split_rule = (max_feature, max_thresh), is_leaf=0, label=None)
                if verbose == True:
                    print(repr(node))
                return node


    def iter_thresh(self, X, y):
        row_f = sorted(set(X))
        col_l = set(y)
        freq_matrix = np.zeros([len(row_f), len(col_l)])
        for i, j in enumerate(row_f):
            for k, l in enumerate(col_l):
                freq_matrix[i, k] = len(y[np.where(y[np.where(X==j)]==l)])
        all_thresh = np.array(row_f[1:] + row_f[-1:]) / 2.
        left_freq = np.zeros([len(col_l)])
        right_freq = np.sum(freq_matrix, axis=0)
        left_freq_sum = 0
        max_thresh = all_thresh[0]

        max_gain = self.information_gain(left_freq, right_freq)
        for i, thresh in enumerate(all_thresh):
            left_freq += freq_matrix[i, :]
            right_freq -= freq_matrix[i, :]
            gain = self.information_gain(left_freq, right_freq)
            if gain > max_gain:
                max_gain = gain
                max_thresh = thresh
        return max_thresh, max_gain


    def segmenter(self, X, y, random_f = -1):
        """
        TODO: compute entropy gain for all single-dimension splits,
        return the feature and the threshold for the split that
        has maximum gain
        """
        x = X.shape[1]
        if random_f == -1:
            all_features = np.arange(x)
        else:
            all_features = np.random.choice(range(x), random_f)

        all_features = np.arange(x)
        max_gain = 1e-10
        max_thresh = 0
        max_feature = 0
        for i in all_features:
            thresh, gain = self.iter_thresh(X[:, i], y)
            if gain > max_gain:
                max_gain = gain
                max_thresh = thresh
```

```
                max_feature = i
        return max_feature, max_thresh


    def fit(self, X, y, random_f = -1, verbose = False):
        """
        TODO: fit the model to a training set. Think about what would be
        your stopping criteria
        """
        self.data = X
        self.labels = y
        S = np.array(range(len(y)))
        self.root = self.split(S, 1 , random_f = random_f,verbose = verbose)

        return self


    def predict(self, X, T = 0):
        """
        TODO: predict the labels for input data
        """
        if T == 1: # T = 1 for random forest
            X = np.reshape(X, [1, len(X)])
            row_num = 1
        else:
            row_num = X.shape[0]
        labels = np.zeros(row_num)
        depth = 0
        for i in range(row_num):
            current_node = self.root
            while current_node.is_leaf == 0:
                feature = current_node.split_rule[0]
                thresh = current_node.split_rule[1]
                if X[i,:][feature] <= thresh:
                    current_node = current_node.left
                else:
                    current_node = current_node.right
            depth += 1
            labels[i] = current_node.label
        return labels

    def accuracy(self, X, y_val, T = 0):
        y_pred = self.predict(X, T = T)
        len_y = float(len(y_pred))
        return np.sum(y_pred == y_val) / len_y


class RandomForest():

    def __init__(self, n_trees=20, n_sample=1000, random_f=-1, max_depth=200):
        """
        TODO: initialization of a random forest
        """
        self.n_trees = n_trees
        self.n_sample = n_sample
        self.random_f = random_f
        self.max_depth = max_depth
        self.trees = np.array([DecisionTree(max_depth)] * n_trees)

    def fit(self, X, y, verbose = False):
        """
```

```
        TODO: fit the model to a training set.
        """
        if self.random_f == -1:
            self.random_f = int(np.sqrt(X.shape[1]))
        results = np.zeros(self.n_trees, dtype=object)
        for i, dt in enumerate(self.trees):
            print('#%d. tree' % (i+1))
            idx = np.random.choice(range(len(X)), self.n_sample)
            sub_X = X[idx, :]
            sub_y = y[idx]
            results[i] = dt.fit(sub_X, sub_y, random_f=self.random_f, verbose = verbose
)
        self.trees = results

    def predict(self, X, T = 1):
        """
        TODO: predict the labels for input data
        """
        row_num = X.shape[0]
        labels = []
        for i in range(row_num):
            pred = np.zeros(self.n_trees)
            for j, dt in enumerate(self.trees):
                pred[j] = dt.predict(X[i, :], T = T)
            labels.append(dt.max_count(pred))
        return labels

    def accuracy(self, X, y_val, T = 1):
        y_pred = self.predict(X,T = T)
        N = float(len(y_pred))
        return np.sum(y_pred == y_val) / N
```

In [97]:

```
# Load titanic data
titanic_train = pd.read_csv('datasets/titanic/titanic_training.csv')
titanic_test = pd.read_csv('datasets/titanic/titanic_testing_data.csv')

# remove ticket and cabin column
titanic_train.drop(['ticket', 'cabin'], inplace=True, axis=1)
titanic_test.drop(['ticket', 'cabin'], inplace=True, axis=1)
```

In [ ]:

```

```

In [ ]:

```

```

In [100]:

```
# Load titanic data
titanic_train = pd.read_csv('datasets/titanic/titanic_training.csv')
titanic_test = pd.read_csv('datasets/titanic/titanic_testing_data.csv')

# remove ticket and cabin column
titanic_train.drop(['ticket', 'cabin'], inplace=True, axis=1)
titanic_test.drop(['ticket', 'cabin'], inplace=True, axis=1)

# fillna
titanic_train["survived"].fillna(titanic_train["survived"].mode(), inplace=True)
for data in [titanic_train, titanic_test]:
    data["fare"].fillna(data["fare"].median(), inplace=True)
    data["embarked"].fillna(data["embarked"].mode(), inplace=True)
    data["parch"].fillna(data["parch"].median(), inplace=True)
    data["sibsp"].fillna(data["sibsp"].median(), inplace=True)
#     data["age"].fillna(data["age"].median(), inplace=True)
    data["sex"].fillna(data["sex"].mode(), inplace=True)
    data["pclass"].fillna(data["pclass"].median(), inplace=True)
```

In [82]:

```
# feature engineering
```

In [102]:

```
titanic_train['family_size'] = titanic_train['sibsp'] + titanic_train['parch']
titanic_test['family_size'] = titanic_test['sibsp'] + titanic_test['parch']
```

In [85]:

```
# t_label_enc['survived'].fillna(t_label_enc['survived'].mode(),inplace = True)
```

In [103]:

```
titanic_train['family'] = ''
titanic_train.loc[titanic_train['family_size'] == 0, 'family'] = 0
titanic_train.loc[(titanic_train['family_size'] > 0) & (titanic_train['family_size'] <=
3), 'family'] = 1
titanic_train.loc[(titanic_train['family_size'] > 3) & (titanic_train['family_size'] <=
6), 'family'] = 2
titanic_train.loc[titanic_train['family_size'] > 6, 'family'] = 3

titanic_test['family'] = ''
titanic_test.loc[titanic_test['family_size'] == 0, 'family'] = 0
titanic_test.loc[(titanic_test['family_size'] > 0) & (titanic_test['family_size'] <= 3
), 'family'] = 1
titanic_test.loc[(titanic_test['family_size'] > 3) & (titanic_test['family_size'] <= 6
), 'family'] = 2
titanic_test.loc[titanic_test['family_size'] > 6, 'family'] = 3
```

In [104]:

```
titanic_train['ageRange'] = pd.cut(titanic_train['age'], bins=[0,15,35,45,60,200], labe
ls=['<15','15-35','35-45','40-60','>60'], include_lowest=True)
titanic_train['fareRange'] = pd.cut(titanic_train['fare'], bins=[0,10,30,60,1000], labe
ls=['<10','10-30','30-60','>60'], include_lowest=True)
titanic_test['ageRange'] = pd.cut(titanic_test['age'], bins=[0,15,35,45,60,200], labels
=['<15','15-35','35-45','40-60','>60'], include_lowest=True)
titanic_test['fareRange'] = pd.cut(titanic_test['fare'], bins=[0,10,30,60,1000], labels
=['<10','10-30','30-60','>60'], include_lowest=True)
```

In [88]:

```
new = titanic_train[['ageRange','family']].groupby(titanic_train['ageRange']).mean()
new
# plt.plot(titanic_train['ageRange'],titanic_train['family_size'])
```

Out[88]:

	family
ageRange	
<15	1.325000
15-35	0.376321
35-45	0.593750
40-60	0.537634
>60	0.433333

In [31]:

```
titanic_train.head()
```

Out[31]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	family_size	family	ageI
0	0.0	3.0	male	NaN	0.0	0.0	8.0500	S	0.0	0	
1	0.0	1.0	male	22.0	0.0	0.0	135.6333	C	0.0	0	
2	0.0	2.0	male	23.0	0.0	0.0	15.0458	C	0.0	0	
3	0.0	2.0	male	42.0	0.0	0.0	13.0000	S	0.0	0	
4	0.0	3.0	male	20.0	0.0	0.0	9.8458	S	0.0	0	

In [35]:

```
titanic_train['family'].unique()
```

Out[35]:

```
array([0, 1, 2, 3], dtype=int64)
```

In [108]:

```
sub_zero = titanic_train[titanic_train['family'] == 0]
sub_one = titanic_train[titanic_train['family'] == 1]
sub_two = titanic_train[titanic_train['family'] == 2]
sub_three = titanic_train[titanic_train['family'] == 3]
```

In [109]:

```
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
sub_zero_fill = imp.fit_transform(sub_zero[['age','family']])
sub_one_fill = imp.fit_transform(sub_one[['age','family']])
sub_two_fill = imp.fit_transform(sub_two[['age','family']])
sub_three_fill = imp.fit_transform(sub_three[['age','family']])

new = np.vstack((sub_zero_fill,sub_one_fill,sub_two_fill,sub_three_fill))
titanic_train['age_new'] = new[:,0]
titanic_train['family_new'] = new[:,1]
titanic_train.head()
```

```
C:\Users\dizhe\Anaconda3\envs\mcm\lib\site-packages\sklearn\utils\deprecat
ion.py:58: DeprecationWarning: Class Imputer is deprecated; Imputer was de
precated in version 0.20 and will be removed in 0.22. Import impute.Simple
Imputer from sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

Out[109]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	family_size	family	agel
0	0.0	3.0	male	NaN	0.0	0.0	8.0500	S	0.0	0	
1	0.0	1.0	male	22.0	0.0	0.0	135.6333	C	0.0	0	
2	0.0	2.0	male	23.0	0.0	0.0	15.0458	C	0.0	0	
3	0.0	2.0	male	42.0	0.0	0.0	13.0000	S	0.0	0	
4	0.0	3.0	male	20.0	0.0	0.0	9.8458	S	0.0	0	

In [175]:

```
sub_zero = titanic_test[titanic_test['family'] == 0]
sub_one = titanic_test[titanic_test['family'] == 1]
sub_two = titanic_test[titanic_test['family'] == 2]
sub_three = titanic_test[titanic_test['family'] == 3]
```

In [176]:

```
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
sub_zero_fill = imp.fit_transform(sub_zero[['age','family']])
sub_one_fill = imp.fit_transform(sub_one[['age','family']])
sub_two_fill = imp.fit_transform(sub_two[['age','family']])
sub_three_fill = imp.fit_transform(sub_three[['age','family']])

new = np.vstack((sub_zero_fill,sub_one_fill,sub_two_fill,sub_three_fill))
titanic_test['age_new'] = new[:,0]
titanic_test['family_new'] = new[:,1]
titanic_test.head()
```

```
C:\Users\dizhe\Anaconda3\envs\mcm\lib\site-packages\sklearn\utils\deprecat
ion.py:58: DeprecationWarning: Class Imputer is deprecated; Imputer was de
precated in version 0.20 and will be removed in 0.22. Import impute.Simple
Imputer from sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

Out[176]:

	pclass	sex	age	sibsp	parch	fare	embarked	family_size	family	ageRange	fa
0	1.0	female	24.0	0.0	0.0	69.3000	C	0.0	0	15-35	
1	1.0	female	44.0	0.0	1.0	57.9792	C	1.0	1	35-45	
2	3.0	male	1.0	5.0	2.0	46.9000	S	7.0	3	<15	
3	3.0	male	29.0	0.0	0.0	7.8750	S	0.0	0	15-35	
4	2.0	male	30.0	0.0	0.0	13.0000	S	0.0	0	15-35	

In [ ]:

In [177]:

```
titanic_train['ageRange'] = pd.cut(titanic_train['age'], bins=[0,15,35,45,60,200], labe
ls=['<15','15-35','35-45','40-60','>60'], include_lowest=True)
titanic_train['fareRange'] = pd.cut(titanic_train['fare'], bins=[0,10,30,60,1000], labe
ls=['<10','10-30','30-60','>60'], include_lowest=True)
titanic_test['ageRange'] = pd.cut(titanic_test['age'], bins=[0,15,35,45,60,200], labels
=['<15','15-35','35-45','40-60','>60'], include_lowest=True)
titanic_test['fareRange'] = pd.cut(titanic_test['fare'], bins=[0,10,30,60,1000], labels
=['<10','10-30','30-60','>60'], include_lowest=True)
```

In [178]:

```
# get dummy
t_train = titanic_train.copy()
dummy_col = ['pclass','sex','embarked','ageRange','fareRange']
for col in dummy_col:
    dummy = pd.get_dummies(t_train[col],drop_first=True)
    t_train = pd.concat([t_train,dummy], axis = 1)
t_train.head()

t_test = titanic_test.copy()
for col in dummy_col:
    dummy = pd.get_dummies(t_test[col],drop_first=True)
    t_test = pd.concat([t_test,dummy], axis = 1)
t_test.head()
```

Out[178]:

	pclass	sex	age	sibsp	parch	fare	embarked	family_size	family	ageRange	...
0	1.0	female	24.0	0.0	0.0	69.3000	C	0.0	0	15-35	...
1	1.0	female	44.0	0.0	1.0	57.9792	C	1.0	1	35-45	...
2	3.0	male	1.0	5.0	2.0	46.9000	S	7.0	3	<15	...
3	3.0	male	29.0	0.0	0.0	7.8750	S	0.0	0	15-35	...
4	2.0	male	30.0	0.0	0.0	13.0000	S	0.0	0	15-35	...

5 rows × 25 columns

In [183]:

```
t_test.columns
```

Out[183]:

```
Index([    'age_new', 'family_new',           2.0,           3.0,        'mal
e',
                 'Q',            'S',       '15-35',       '35-45',       '40-6
0',
               '>60',       '10-30',       '30-60',         '>60'],
      dtype='object')
```

In [181]:

```
t_train.drop(['sex','age','sibsp','parch','fare','embarked','pclass','family_size','age
Range','fareRange','family'], inplace=True, axis=1)
t_test.drop(['sex','age','sibsp','parch','fare','embarked','pclass','family_size','ageR
ange','fareRange','family'], inplace=True, axis=1)
```

In [184]:

```
t_label_enc = t_train[['survived']]
t_train_enc = t_train.iloc[:,1:]
t_test_enc = t_test
t_train_enc.head()
```

Out[184]:

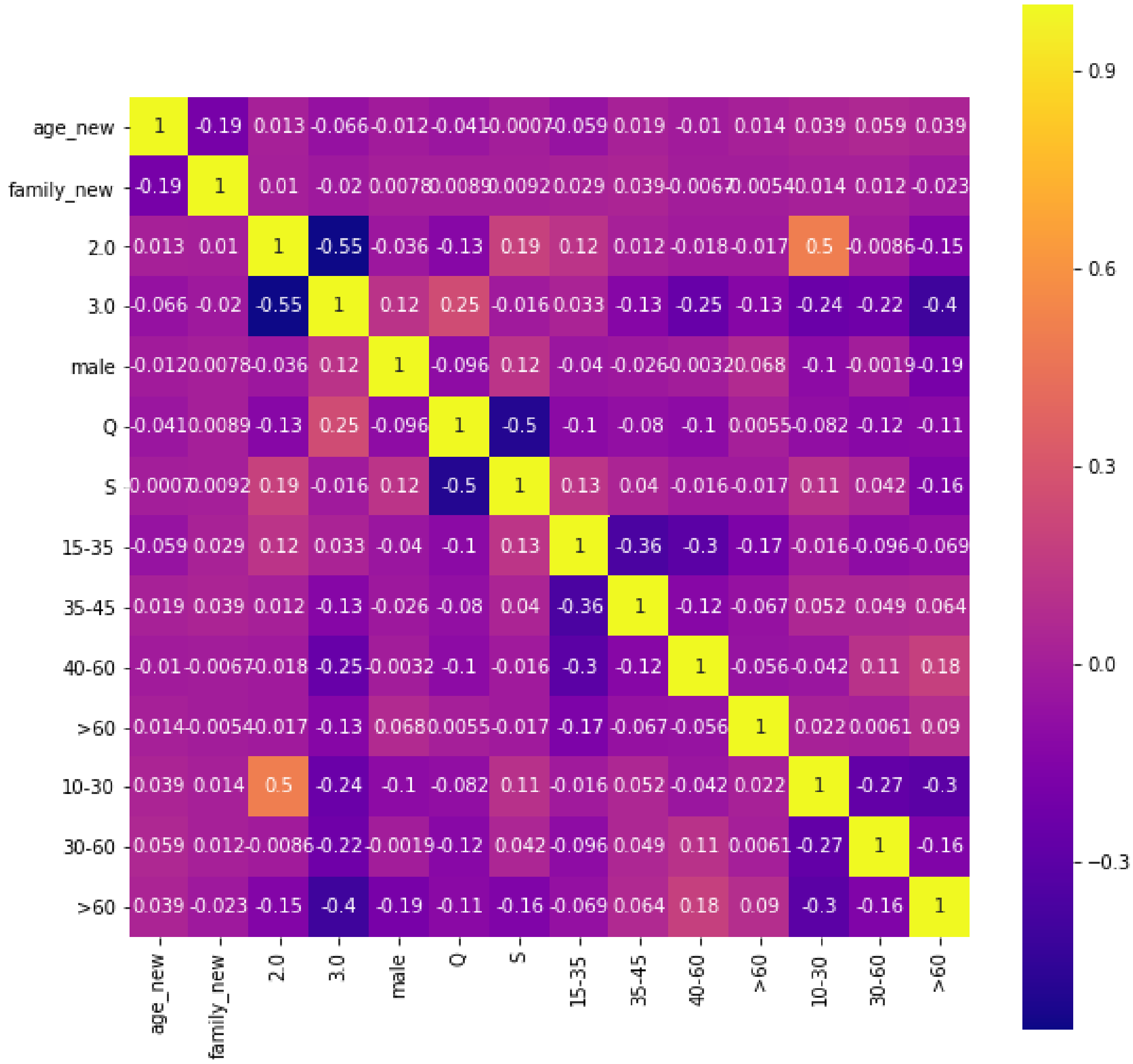
	age_new	family_new	2.0	3.0	male	Q	S	15-35	35-45	40-60	>60	10-30	30-60	>60
0	31.828375	0.0	0	1	1	0	1	0	0	0	0	0	0	0
1	22.000000	0.0	0	0	1	0	0	1	0	0	0	0	0	1
2	23.000000	0.0	1	0	1	0	0	1	0	0	0	1	0	0
3	42.000000	0.0	1	0	1	0	1	0	1	0	0	1	0	0
4	20.000000	0.0	0	1	1	0	1	1	0	0	0	0	0	0

In [185]:

```
# check the correlation
import seaborn as sns
hotmap = plt.cm.plasma
plt.figure(figsize=(10,10))
sns.heatmap(t_train_enc.corr(),square=True, cmap=hotmap,annot = True)
```

Out[185]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x14c388de240>
```


age_new
family_new
2.0
3.0
male
Q
S
15-35
35-45
40-60
>60
10-30
30-60
>60
0.9
0.6
0.3
0.0
−0.3

In [108]:

```
# from sklearn.feature_selection import SelectKBest, chi2

# #Feature selection using SelectKBest
# test = SelectKBest(score_func=chi2, k=8)
# fit = test.fit(t_train_enc, t_label_enc)
# # summarize scores
# np.set_printoptions(precision=3)
# print(fit.scores_)
# features = fit.transform(t_train_enc)
# # summarize selected features
# print(features[0:5,:])
# print (t_train_enc.head())
```

In [186]:

```
t_label_enc.fillna(0,inplace = True)
t_label_enc.isnull().sum()
```

```
C:\Users\dizhe\Anaconda3\envs\mcm\lib\site-packages\pandas\core\frame.py:4
034: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
  downcast=downcast, **kwargs)
```

Out[186]:

```
survived    0
dtype: int64
```

In [187]:

```
t_label_enc.isnull().sum()
```

Out[187]:

```
survived    0
dtype: int64
```

In [189]:

```
from sklearn.feature_selection import SelectKBest, chi2
best = SelectKBest(score_func=chi2, k=14)
fit = best.fit(t_train_enc,t_label_enc)

dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(pd.DataFrame(t_train_enc).columns)

featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']
topindex = featureScores['Score'].sort_values(ascending = False)[:10].index
topindex = topindex.tolist()

train_X_selected = t_train_enc.iloc[:,topindex]
test_X_selected = t_test_enc.iloc[:,topindex]
```

In [ ]:

```

```

In [196]:

```
t_train_sele = t_train_enc#[['age','fare',1.0,'female','male']]
t_test_sele = t_test_enc#[['age','fare',1.0,'female','male']]
t_train_sele = t_train_sele.values
t_test_sele = t_test_sele.values
```

In [197]:

```
# split data
idx = np.random.choice(range(len(t_label_enc)),len(t_label_enc),replace = False)
split = int(len(t_label_enc)*0.8)
t_train_sele[idx][:split,]
X_train,X_val = t_train_sele[idx][:split,],t_train_sele[idx][split:,]
y_train,y_val = np.array(t_label_enc)[idx][:split],np.array(t_label_enc)[idx][split:]

y_val = y_val.astype("int64")
y_val = y_val.flatten()
y_train = y_train.astype("int64")
y_train = y_train.flatten()
```

In [198]:

```
# try training
dt = DecisionTree(3)
dt.fit(X_train, y_train, verbose = True)
train_acc = dt.accuracy(X_train, y_train)
print('Training accuracy: ', train_acc)
val_acc = dt.accuracy(X_val, y_val)
print('Validation accuracy: ', val_acc)
```

```
[Feature: 4, Threshold: <=0.5]
        [Feature: 3, Threshold: <=0.5]
                (Therefore the person was: survived)
                (Therefore the person was: survived)
        [Feature: 3, Threshold: >0.5]
                (Therefore the person was: died)
                (Therefore the person was: died)
Training accuracy:  0.78375
Validation accuracy:  0.805
```

In [199]:

```
y_pred = dt.predict(t_test_sele)
y_pred = np.array(y_pred)
```

In [200]:

```
y_pred.shape
```

Out[200]:

```
(310,)
```

In [201]:

```
# save
def results_to_csv(y_test):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1  # Ensures that the index starts at 1.
    df.to_csv('submission_titanic_rf_7.29.23.12.csv', index_label='Id')

results_to_csv(y_pred)
```

1.7 Kaggle

team name: JoKer
spam score:85.197
titanic score: 75.483

I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted.

Di Zhen

Scanned by CamScanner