CSCE633 Homework 04

Lu Sun

October 2020

Question: Decision Tree and Random Forest

(1) Data exploration:

- Plot the histograms of the training data
- Compute the numbers of samples belonging to the benign or the malignant case respectively.

Answer:
The code is as follow:

```
import matplotlib.pyplot as plt
import numpy as np

NameMap = {1:'Clump Thickness',2:'Uniformity of Cell Size',
           3:'Uniformity of Cell Shape',4:'Marginal Adhesion',
           5:'Single Epithelial Cell Size',6:'Bare Nuclei',
           7:'Bland Chromatin',8:'Normal Nucleoli',
           9:'Mitoses',10:'Class'}

################################plot################################
def funcPlot(data_set):
  plt.style.use('ggplot')
  fig = plt.figure(figsize=[25,25])
  for i in range(10):
    a = fig.add_subplot(4,3,i+1)
    x = data_set[0::1,i]
    _ = plt.hist(x, bins=np.linspace(0.5,10.5,11),
                 color = 'w',ec = 'darkblue',lw=1.5)

    if i !=9:
      plt.xlabel('discrete values: {1,...,10}')
    else:
      plt.xlabel('2 for benign, 4 for malignant')
    plt.ylabel('Number of train data')
    a.set_title(NameMap[i+1])
  plt.show()

funcPlot(train_data)

```

```
######################count number#################################
unique, count = np.unique(train_data[0::1,-1],
                          return_counts=True, axis=0)
print('n_benign:{},n_malign:{},n_tol:{}'.format(count[0],
                                                count[1],
                                                train_data.shape[0]))
```

The results are shown as follow:

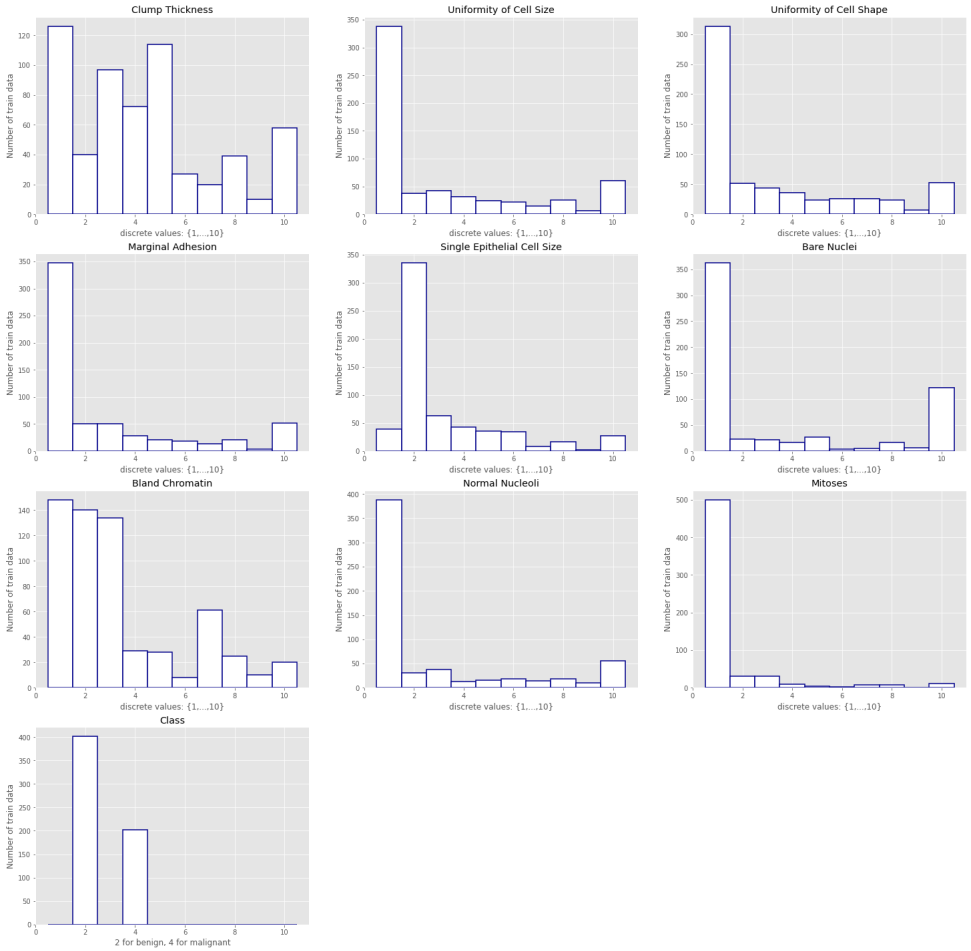

Figure 1: Histograms of the training data

benign	malignant
401	202

Table 1: Count the number of samples per class in the training data.

(2) Conditional entropy:

- Compute the conditional entropies for each feature.
- Which features are the most discriminative of the outcome.

Answer:
Two different method are used here to generate the conditional entropy.

- Method 1:

$$\begin{aligned} H(Y|X) &= \sum_{i=1}^{10} P(X=i)H(Y|X=i) \\ &= \sum_{i=1}^{10} P(X=i)(-\sum_{j=2,4} P(Y=j|X=i)\log(P(Y=j|X=i))) \end{aligned}$$

- Method 2:

$$\begin{aligned} H(Y|X) &= H(Y,X) - H(X) \\ &= -\sum_{j=2,4}\sum_{i=1}^{10} P(Y=j,X=i)\log(P(Y=j,X=i)) - [-\sum_{j=2,4} P(Y=j)\log(P(Y=j))] \end{aligned}$$

The code is as follow:

Listing 1: Method 1 conditional entropy

```
##Conditional Entropy
def cEntropy1(Y,X):
  XY = np.c_[X,Y]
  unique, count = np.unique(XY, return_counts=True, axis=0)
  total_num = sum(count)

  condtion_entorpy = np.zeros(10)
  for i in range(10):
    index = np.nonzero(unique[0::1,0]==i+1)
    if index[0].size !=0:
      x_subnum = sum(count[index])
      #print(i+1,index[0])
      Pyx = count[index]/x_subnum
      #print(count[index]/sum(count[index]))
      #print(Pyx.dot(np.log(Pyx).T),sum(count[index])/sum(count))
      condtion_entorpy[i] = -Pyx.dot(np.log(Pyx).T)* ...
          (x_subnum/total_num)
  return sum(condtion_entorpy)
```

Listing 2: Method 2 conditional entropy

```
##Entropy
def entropy(Y):
    """
    Also known as Shanon Entropy
```

```
    Reference: ...
        https://en.wikipedia.org/wiki/Entropy_(information_theory)
    """
    unique, count = np.unique(Y, return_counts=True, axis=0)
    prob = count/len(Y)
    en = np.sum((-1)*prob*np.log(prob))
    return en


#Joint Entropy
def jEntropy(Y,X):
    """
    H(Y;X)
    Reference: https://en.wikipedia.org/wiki/Joint_entropy
    """
    YX = np.c_[Y,X]
    return entropy(YX)

#Conditional Entropy
def cEntropy2(Y, X):
    """
    conditional entropy = Joint Entropy - Entropy of X
    H(Y|X) = H(Y;X) - H(X)
    Reference: https://en.wikipedia.org/wiki/Conditional_entropy
    """
    return jEntropy(Y, X) - entropy(X)
```

The code for computing the conditional entropies is as follow:

```
Y = train_data[0::1,-1]
#Method1
ce1 = np.zeros(9)
#Method2
ce2 = np.zeros(9)
for i in range(9):
  X = train_data[0::1,i]
  ce1[i] = cEntropy1(Y,X)
  ce2[i] = cEntropy2(Y, X)
print(ce1)
print(ce2)
#Get top 3 most discriminative features
#print(NameMap[ce1.argmin()+1],ce1.min())
min3_index = ce1.argsort()[0:3]
for ind in min3_index:
  print(NameMap[ind+1],ce1[ind])
```

The results are shown in the following table: Among the 9 features, the top

Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape
0.32849172	0.14602451	0.1620714
Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei
0.2941173	0.26541363	0.19095501
Bland Chromatin	Normal Nucleoli	Mitoses
0.23870233	0.30680583	0.48587901

Table 2: conditional entropies for each feature

3 most discriminative of the outcome features are in the following table:

features	conditional entropy
Uniformity of Cell Size	0.14602451066410996
Uniformity of Cell Shape	0.1620714007513602
Bare Nuclei	0.1909550109027593

Table 3: The most discriminative features

(3) Decision tree classification:

- Tune hyper-parameters (e.g., tree depth) by 5-fold cross-validation on the training set.
- Report accuracy for all hyper-parameters when training.
- Report accuracy for testing with the best hyper-parameter.

Answer:

The code for tuning hyper-parameters are shown as follow:

```
from sklearn import tree
from sklearn.model_selection import cross_validate

#X data
X = train_data[0::1,0:9]
#Y data
Y = train_data[0::1,-1]

#accuracy of all 5 folds
cv_results = np.zeros((20,5))
#average accuracy of 5 folds
acc_ave = np.zeros(20)

#Tune Process
#tree depth: {1,2,3,...,20}
max_depth = np.linspace(1,20,20).astype(int)
for ind,md in enumerate(max_depth):
  #define tree_depth=md Decision Tree
  clf = tree.DecisionTreeClassifier(max_depth=md)
  #5-fold cross_validation with accuracy as socring
  cv_results[ind,:] = cross_validate(clf, X, Y,
                                     scoring='accuracy',
                                     cv=5)['test_score']
  #print(np.mean(cv_results['test_score']))
  #print(cv_results)
  acc_ave[ind] = np.mean(cv_results[ind])
best_depth = max_depth[acc_ave.argmax()]
```

According to the following table 4, the best hyper-parameters are shown as follow: The accuracy for all hyper-parameters when trainig is shown as follow:

Tree Depth:	2
Ave Acc:	0.9635537190082644

Tree Depth	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	AVE
1	0.90909091	0.95041322	0.90909091	0.96666667	0.89166667	0.92538567
2	0.95867769	0.94214876	0.96694215	0.975	0.975	0.96355372
3	0.94214876	0.9338843	0.95041322	0.99166667	0.975	0.95862259
4	0.96694215	0.9338843	0.95041322	0.975	0.975	0.96024793
5	0.96694215	0.95041322	0.94214876	0.975	0.96666667	0.96023416
6	0.92561983	0.94214876	0.90909091	0.975	0.96666667	0.94370523
7	0.95867769	0.9338843	0.90909091	0.98333333	0.93333333	0.94366391
8	0.94214876	0.94214876	0.91735537	0.975	0.93333333	0.94199725
9	0.95867769	0.94214876	0.91735537	0.975	0.96666667	0.9519697
10	0.9338843	0.9338843	0.90909091	0.98333333	0.96666667	0.9453719
11	0.96694215	0.94214876	0.90909091	0.96666667	0.93333333	0.94363636
12	0.95041322	0.94214876	0.9338843	0.98333333	0.96666667	0.95528926
13	0.94214876	0.9338843	0.91735537	0.975	0.975	0.94867769
14	0.94214876	0.9338843	0.90909091	0.96666667	0.94166667	0.93869146
15	0.95867769	0.94214876	0.91735537	0.98333333	0.96666667	0.95363636
16	0.95867769	0.9338843	0.90909091	0.96666667	0.93333333	0.94033058
17	0.95041322	0.94214876	0.90909091	0.975	0.975	0.95033058
18	0.94214876	0.94214876	0.91735537	0.96666667	0.94166667	0.94199725
19	0.95867769	0.9338843	0.92561983	0.96666667	0.96666667	0.95030303
20	0.9338843	0.94214876	0.92561983	0.96666667	0.94166667	0.94199725

Table 4: Accuracy for all hyper-parameters

The code for testint is shown as follow:

```
X_train = train_data[0::1,0:9]
Y_train = train_data[0::1,-1]
X_test = test_data[0::1,0:9]
Y_test = test_data[0::1,-1]

acc1 = np.zeros(100)
for i in range(len(acc1)):
  clf = tree.DecisionTreeClassifier(max_depth=2)
  clf.fit(X_train,Y_train)
  acc1[i] = clf.score(X_test,Y_test)
print(np.mean(acc1),np.std(acc1))
```

The testing resutls are in the following table:

Tree Depth	2
Test Acc	0.8875(0)

Table 5: Testing Results

(4) Random forest tree classification:

- Tune hyper-parameters (e.g., tree depth, number of trees) by 5-fold cross-validation on the training set.
- Report accuracy for all hyper-parameters when training.
- Report accuracy for testing with the best hyper-parameter.
- Compare and contrast the performance of the decision tree with the random forest classifier

Answer:

The code for tuning hyper-parameters are shown as follow:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_validate
import numpy as np

X = train_data[0::1,0:9]
Y = train_data[0::1,-1]

best_ne_md = np.zeros(2,dtype=int)
cv_results = np.zeros((5,6,5))
acc_ave = np.zeros((5,6))

n_estimators = np.linspace(10,30,5).astype(int)#number of trees
max_depth = np.linspace(3,8,6).astype(int)     #depth of trees

#Tune
for jnd,ne in enumerate(n_estimators):
  for ind,md in enumerate(max_depth):
    clf = RandomForestClassifier(n_estimators=ne,max_depth=md)
    cv_results[jnd,ind,:] = cross_validate(clf, X, Y,
                                           scoring='accuracy',
                                           cv=5)['test_score']
    print(jnd,ind,cv_results[jnd,ind,:])
    #print(cv_results)
    acc_ave[jnd,ind] = np.mean(cv_results[jnd,ind])

from numpy import unravel_index
ne,md = unravel_index(acc_ave.argmax(), acc_ave.shape)
best_ne_md = [n_estimators[ne],max_depth[md]]
```

According to the following table 6, the best hyper-parameters are shown as follow:

Tree Depth:	7
Number of Trees:	15
Ave Acc:	0.9751652892561984

The accuracy for all hyper-parameters when trainig is shown as follow:

# Trees	Tree Depth	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	AVE
10	3	0.966942	0.92562	0.975207	0.983333	0.991667	0.968554
10	4	0.983471	0.933884	0.958678	0.991667	1.0	0.97354
10	5	0.958678	0.942149	0.966942	0.983333	0.991667	0.968554
10	6	0.958678	0.942149	0.958678	0.975	1.0	0.966901
10	7	0.966942	0.92562	0.933884	0.983333	1.0	0.961956
10	8	0.950413	0.942149	0.942149	0.975	0.983333	0.958609
15	3	0.958678	0.92562	0.966942	0.983333	0.983333	0.963581
15	4	0.975207	0.933884	0.975207	0.983333	0.991667	0.97186
15	5	0.975207	0.942149	0.950413	0.983333	1.0	0.97022
15	6	0.966942	0.942149	0.958678	0.983333	1.0	0.97022
15	7	0.983471	0.942149	0.975207	0.983333	0.991667	0.975165
15	8	0.966942	0.942149	0.958678	0.975	1.0	0.968554
20	3	0.966942	0.933884	0.958678	0.983333	1.0	0.968567
20	4	0.975207	0.933884	0.958678	0.983333	1.0	0.97022
20	5	0.975207	0.942149	0.966942	0.983333	0.991667	0.97186
20	6	0.975207	0.933884	0.958678	0.983333	0.983333	0.966887
20	7	0.975207	0.933884	0.958678	0.983333	1.0	0.97022
20	8	0.966942	0.933884	0.950413	0.991667	1.0	0.968581
25	3	0.975207	0.933884	0.966942	0.983333	0.991667	0.970207
25	4	0.975207	0.942149	0.966942	0.983333	0.991667	0.97186
25	5	0.975207	0.933884	0.966942	0.983333	0.991667	0.970207
25	6	0.975207	0.942149	0.958678	0.991667	1.0	0.97354
25	7	0.958678	0.933884	0.975207	0.983333	0.991667	0.968554
25	8	0.958678	0.933884	0.958678	0.983333	1.0	0.966915
30	3	0.966942	0.92562	0.966942	0.983333	1.0	0.968567
30	4	0.966942	0.933884	0.966942	0.983333	1.0	0.97022
30	5	0.983471	0.933884	0.966942	0.983333	0.991667	0.97186
30	6	0.983471	0.933884	0.958678	0.983333	1.0	0.971873
30	7	0.975207	0.942149	0.958678	0.983333	1.0	0.971873
30	8	0.958678	0.933884	0.975207	0.983333	1.0	0.97022

Table 6: Accuracy for all hyper-parameters

The code for testint is shown as follow:

```
X_train = train_data[0::1,0:9]
Y_train = train_data[0::1,-1]
X_test = test_data[0::1,0:9]
Y_test = test_data[0::1,-1]

acc2 = np.zeros(100)
for i in range(len(acc2)):
  clf = RandomForestClassifier(n_estimators=best_ne_md[0],
                               max_depth=best_ne_md[1])
  clf.fit(X_train,Y_train)
  acc2[i] = clf.score(X_test,Y_test)
print(np.mean(acc2),np.std(acc2))
```

The testing resutls are in the following table: The performance of decision tree

Tree Depth	7
Number of Trees	15
Test Acc	0.9051250000000002(0.019692558873848777)

Table 7: Testing Results

and random forest classifier is as follow: Through the above table, the following

	Test Acc Mean	Test ACC std
Decision Tree	0.8875	0
Random Forest	0.9051250000000002	0.019692558873848777

Table 8: Comparison

conclusion can be obtained:

- Random Forest average performance on testing data set is better than Decision Tree
- Decision Tree is more stable than Random Forest whose Test ACC std =0.