Homework 1 - Raul G. Martinez (PID: A12461871)

DSE 220: Machine Learning

Due Date : April 16, 11:59 PM

1. Instructions

The answers to the questions should be submitted on Gradescope with the codes. You don't need to explain your approach (unless specified) so please be concise in your Gradescope submission. To obtain full marks for a question, both the answer and the code should be correct. Completely wrong (or missing) code with correct answer will result in zero marks. Please make sure that your code is clean and well segmented for each question.

2. Data

Download the 'wine modified' and 'wine' (train, validation and test) data from the github. Use the 'wine modified' data for the data preprocessing section and the wine train, validation and test data for the other two sections.

In [149]:
```
# import libraries
import pandas as pd
import numpy as np
```

```
In [150]: # read csv file
          df = pd.read_csv('wine_modified.csv')

          print(df.shape)
          df.head()
```

```
(178, 14)
```

Out[150]:

	class	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315	Proline
0	1.0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	1.0	13.20	1.78	NaN	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	1.0	13.16	2.36	NaN	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
3	1.0	14.37	NaN	2.50	NaN	NaN	3.85	NaN	NaN	NaN	7.80	NaN	NaN	NaN
4	1.0	13.24	2.59	NaN	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0

3. Data Preprocessing

The questions in this section are sequential steps. So use the data obtained after Question 1 for Question 2 and so on.

Question 1: Remove the rows with missing labels ('class') and rows with more than 7 missing features. Report the remaining number of rows. (1 mark)

In [151]:
```
# remove rows with missing labels
df.dropna(subset = ['class'], inplace = True)
print('Removed rows with missing labels, Remaining Number of rows: {}'.format(len(df)))

# remove rows with more than 7 missing features
missing_values_param = 7 # filter for missing features
thres_missingfeatures = df.shape[1] - missing_values_param - 1 # define threshold for pandas dropna function
df.dropna(inplace = True, thresh = thres_missingfeatures)
print('Removed rows with more than 7 missing features, Remaining Number of rows: {}'.format(len(df)))

df.head()
```

```
Removed rows with missing labels, Remaining Number of rows: 168
Removed rows with more than 7 missing features, Remaining Number of rows: 154
```

Out[151]:

	class	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315	Proline
0	1.0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	1.0	13.20	1.78	NaN	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	1.0	13.16	2.36	NaN	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
4	1.0	13.24	2.59	NaN	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0
5	1.0	14.20	1.76	NaN	15.2	112.0	3.27	NaN	0.34	1.97	6.75	1.05	2.85	1450.0

Question 2: Remove features with > 50% of missing values. For other fea-tures with missing values fill them with the mean of the corresponding features. Report the removed features (if any) and standard deviation of features with missing values after filling. (2 marks)

In [152]:
```python
# remove features with >50% of missing values
df_temp = df.copy() # create copy of old dataframe
thres_missingvalues = len(df)//2 - 1 # find threshold for >50% missing values
df.dropna(axis = 1, thresh = thres_missingvalues, inplace = True)
print('Removed features with >50% missing values: {}'.format(set(df_temp.columns) - set(df.columns)))

# report the mean values for features before filling
print('\nMean values for features before filling:')
print(df[features].apply(lambda x: x.mean()))

# fill other missing values with mean for each feature
print('\nFeatures with missing values: {}'.format(set(df.columns[df.isnull().any()])))
features = [i for i in df.columns if i != 'class']
df = df.apply(lambda x: x.fillna(x.mean()))

# report the standard deviation for features after filling
print('\nStandard deviation for features after filling missing values with mean:')
print(df[features].apply(lambda x: x.std()))

df.head()
```

```
Removed features with >50% missing values: {'Ash'}

Mean values for features before filling:
Alcohol                  12.305935
Malic acid                2.350455
Alcalinity of ash        19.645455
Magnesium                99.496552
Total phenols             2.267403
Flavanoids                1.937983
Nonflavanoid phenols      0.366753
Proanthocyanins           1.582273
Color intensity           4.982338
Hue                       0.954455
OD280/OD315               2.592013
Proline                 727.006494
dtype: float64

Features with missing values: {'Magnesium', 'Flavanoids'}

Standard deviation for features after filling missing values with mean:
Alcohol                   3.804067
Malic acid                1.116005
Alcalinity of ash         3.456794
```

```
Magnesium               14.440377
Total phenols            0.617237
Flavanoids               0.873573
Nonflavanoid phenols     0.127083
Proanthocyanins          0.587671
Color intensity          2.325204
Hue                      0.229412
OD280/OD315              0.723261
Proline                303.033368
dtype: float64
```

Out[152]:

	class	Alcohol	Malic acid	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315	Proline
0	1.0	14.23	1.71	15.6	127.0	2.80	3.060000	0.28	2.29	5.64	1.04	3.92	1065.0
1	1.0	13.20	1.78	11.2	100.0	2.65	2.760000	0.26	1.28	4.38	1.05	3.40	1050.0
2	1.0	13.16	2.36	18.6	101.0	2.80	3.240000	0.30	2.81	5.68	1.03	3.17	1185.0
4	1.0	13.24	2.59	21.0	118.0	2.80	2.690000	0.39	1.82	4.32	1.04	2.93	735.0
5	1.0	14.20	1.76	15.2	112.0	3.27	1.937983	0.34	1.97	6.75	1.05	2.85	1450.0

Question 3: Detect and remove rows with any outliers/incorrect values in fea- tures ’alcohol’ and ’proline’ (if any). Clearly state the basis of your removal. (1 mark)

In [153]:
```python
# detect and remove rows for alcohol

# display negative numbers in alcohol feature
alcohol = np.array(df['Alcohol'])
print('--> Negative numbers present in alcohol feature:')
print(alcohol)

# check for outliers - criteria: 2 std's from the mean to cover about 95% of the data
print('--> Alcohol summary statistics with negative values removed:')
alcohol = alcohol[alcohol>0]
mu = np.mean(alcohol); std = np.std(alcohol); min_ = np.min(alcohol); max_ = np.max(alcohol)
print('mean: {}, std: {}, min: {}, max: {}'.format(mu, std, min_, max_))
print('mean - 2*std: {}, mean + 2*std: {}'.format(mu-2*std, mu+2*std))
print('--> Number of outliers: {}'.format(len(alcohol[(alcohol<mu-2*std) | (alcohol>mu+2*std)])))

print(
"""
Notes:
1. Negative values removed as alcohol content cannot be less than zero.
2. Outlier rows are removed based on 2 std criteria, this gives about 95% coverage for the data.
"""
)

# remove rows for alcohol based on notes from above
df = df[(df['Alcohol']>mu-2*std) & (df['Alcohol']<mu+2*std)]
print('Size of new dataframe: {}'.format(df.shape))
```

```
--> Negative numbers present in alcohol feature:
[14.23  13.2   13.16  13.24  14.2   14.39  14.06  14.83  13.86  14.1
 13.75  14.75  14.38  13.83  14.19  13.64  14.06  12.93  -8.226 12.85
 13.5   13.05  13.39  13.3   13.87  14.02  13.73  13.76  13.51  13.48
 13.28  13.05  14.22  13.41  -8.328 13.24  14.21  14.38  13.9   14.1
 13.05  13.83  13.82  13.77  13.56  14.22  13.72  12.37  12.33  -7.584
 13.67  12.37  12.37  12.37  13.34  12.21  12.29  13.86  13.49  12.99
 11.66  13.03  11.84  12.33  12.7   12.    12.08  13.05  11.84  12.67
 12.16  11.65  11.64  12.08  12.08  12.    12.69  12.29  11.62  12.47
 11.81  12.29  12.37  12.29  12.08  12.6   12.34  11.82  12.51  12.42
 12.25  12.72  11.61  11.46  12.52  11.76  11.41  12.08  11.03  11.82
 12.42  -7.662 11.56  12.42  13.05  11.87  12.07  12.43  11.79  12.37
 12.04  12.86  12.88  12.81  12.7   12.51  12.6   12.25  13.49  12.84
 12.93  13.36  13.52  13.62  12.25  13.16  13.88  13.32  13.5   12.79
 13.11  13.23  12.58  13.17  13.84  12.45  14.34  13.48  12.36  13.69
 12.85  -7.776 13.78  13.73  13.45  13.58  13.4   12.77  14.16  13.71
 13.4   13.27  13.17  14.13 ]
```

```
--> Alcohol summary statistics with negative values removed:
mean: 12.984496644295302, std: 0.8213341873465244, min: 11.03, max: 14.83
mean - 2*std: 11.341828269602253, mean + 2*std: 14.62716501898835
--> Number of outliers: 3

Notes:
1. Negative values removed as alcohol content cannot be less than zero.
2. Outlier rows are removed based on 2 std criteria, this gives about 95% coverage for the data.

Size of new dataframe: (146, 13)
```

```
In [154]: # detect and remove rows for proline

          # check for outliers - criteria: 2 std's from the mean to cover about 95% of the data
          print('--> Proline feature:')
          proline = np.array(df['Proline'])
          print(proline)
          print('--> Proline summary statistics:')
          mu = np.mean(proline); std = np.std(proline); min_ = np.min(proline); max_ = np.max(proline)
          print('mean: {}, std: {}, min: {}, max: {}'.format(mu, std, min_, max_))
          print('mean - 2*std: {}, mean + 2*std: {}'.format(mu-2*std, mu+2*std))
          print('--> Number of outliers: {}'.format(len(proline[(proline<mu-2*std) | (proline>mu+2*std)])))

          print(
          """
          Notes:
          1. Outlier rows are removed based on 2 std criteria, this gives about 95% coverage for the data.
          """
          )

          # remove rows for proline based on notes from above
          df = df[(df['Proline']>mu-2*std) & (df['Proline']<mu+2*std)]
          print('Size of new dataframe: {}'.format(df.shape))
```

```
--> Proline feature:
[1065. 1050. 1185.  735. 1450. 1290. 1295. 1045. 1510. 1320. 1547. 1130.
 1680.  845.  780.  770. 1015.  845.  830. 1195. 1285.  915. 1035. 1285.
 1235. 1095.  920.  880. 1105.  760. 1035.  680. 1080. 1065.  985. 1060.
 1150. 1265. 1190. 1375. 1120.  970. 1285.  520.  680.  630.  420.  678.
  510.  750.  718.  870.  410.  472.  985.  428.  392.  500.  750.  463.
  278.  630.  515.  520.  450.  495.  562.  680.  625.  480.  450.  495.
  290.  345.  937.  625.  428.  660.  406.  710.  562.  438.  415.  672.
  315.  510.  488.  680.  562.  325.  607.  434.  385.  495.  345.  465.
  365.  380.  380.  378.  352.  466.  342.  580.  630.  530.  560.  600.
  650.  695.  720.  580.  590.  600.  780.  520.  550.  855.  830.  415.
  650.  500.  480.  425.  675.  640.  725.  480.  880.  660.  620.  520.
  680.  570.  615.  520.  695.  750.  630.  470.  660.  740.  750.  835.
  840.  560.]
--> Proline summary statistics:
mean: 724.1780821917808, std: 301.03143887448465, min: 278.0, max: 1680.0
mean - 2*std: 122.11520444281155, mean + 2*std: 1326.2409599407501
--> Number of outliers: 5

Notes:
```

```
1. Outlier rows are removed based on 2 std criteria, this gives about 95% coverage for the data.

Size of new dataframe: (141, 13)
```

4. Decision Trees

Note: When predicting for the test data, you should train the model again using train + validation data.

Question 4: Train Decision Tree model on train data for criterions = {'gini', 'entropy'} and report the accuracies on the validation data. Select the best criterion and report the accuracy on the test data. (1 mark)

For information on gini criterion, you can refer: http://statweb.stanford.edu/~jtaylo/courses/stats202/restricted/notes/trees.pdf (http://statweb.stanford.edu/~jtaylo/courses/stats202/restricted/notes/trees.pdf)

In [159]:
```
# import libraries
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import pydotplus
import matplotlib.pyplot as plt
```

In [160]:
```
# read train, validation, and test datasets

x_train = pd.read_csv('wine_train_data.csv')
x_val = pd.read_csv('wine_val_data.csv')
x_test = pd.read_csv('wine_test_data.csv')

y_train = pd.read_csv('wine_train_labels.csv')
y_val = pd.read_csv('wine_val_labels.csv')
y_test = pd.read_csv('wine_test_labels.csv')
```

In [161]:
```python
# train decision tree model on training data and evaluate on validation data

# evaluate criterions = {'gini', 'entropy'}
for c in ['gini', 'entropy']:

    # create model and train
    clf = DecisionTreeClassifier(criterion = c)
    clf.fit(x_train, y_train)

    # make predictions
    y_pred = clf.predict(x_val)
    y_pred_train = clf.predict(x_train)

    # print results
    print('criterion: {}'.format(c))
    print ('\tTrain accuracy = ' + str(np.sum(y_pred_train == np.array(y_train['class']))*1.0/len(y_train)))
    print ('\tValidation accuracy = ' + str(np.sum(y_pred == np.array(y_val['class']))*1.0/len(y_val)))
```

```
criterion: gini
        Train accuracy = 1.0
        Validation accuracy = 0.9487179487179487
criterion: entropy
        Train accuracy = 1.0
        Validation accuracy = 0.9743589743589743
```

In [162]:
```
# select entropy criterion (highest accuracy on val data) and report accuracy on test data

# train the model again using train + validation data
x_train_val = np.concatenate((x_train, x_val))
y_train_val = np.concatenate((y_train['class'], y_val['class']))

# create model and train
clf = DecisionTreeClassifier(criterion = 'entropy')
clf.fit(x_train_val, y_train_val)

# make predictions
y_pred = clf.predict(x_test)
y_pred_train = clf.predict(x_train_val)

# print results
print('criterion: {}'.format('entropy'))
print ('\tTrain + Validation accuracy = ' + str(np.sum(y_pred_train == np.array(y_train_val))*1.0/len(y_train_val)))
print ('\tTest accuracy = ' + str(np.sum(y_pred == np.array(y_test['class']))*1.0/len(y_test)))
```

```
criterion: entropy
        Train + Validation accuracy = 1.0
        Test accuracy = 0.8205128205128205
```

Question 5: Use the criterion selected above to train Decision Tree model on train data for min samples split={2,5,10,20} and report the accuracies on the validation data. Select the best parameter and report the accuracy on the test data. (2 marks)

```
In [167]: # use entropy criterion (higher accuracy on val data) to train model and evaluate on validation data

          # train data for min samples split={2,5,10,20}
          for n in [2, 5, 10, 20]:

              # create model and train
              clf = DecisionTreeClassifier(criterion = 'entropy', min_samples_split = n)
              clf.fit(x_train, y_train)

              # make predictions
              y_pred = clf.predict(x_val)
              y_pred_train = clf.predict(x_train)

              # print results
              print('criterion: {}'.format('entropy'))
              print('min_samples_split: {}'.format(n))
              print ('\tTrain accuracy = ' + str(np.sum(y_pred_train == np.array(y_train['class']))*1.0/len(y_train)))
              print ('\tValidation accuracy = ' + str(np.sum(y_pred == np.array(y_val['class']))*1.0/len(y_val)))
```

```
criterion: entropy
min_samples_split: 2
        Train accuracy = 1.0
        Validation accuracy = 0.9230769230769231
criterion: entropy
min_samples_split: 5
        Train accuracy = 1.0
        Validation accuracy = 0.9743589743589743
criterion: entropy
min_samples_split: 10
        Train accuracy = 0.99
        Validation accuracy = 0.9230769230769231
criterion: entropy
min_samples_split: 20
        Train accuracy = 0.99
        Validation accuracy = 0.9487179487179487
```

In [168]:
```
# select min_samples_split = 5 (highest accuracy on val data) and report accuracy on test data

# train the model again using train + validation data
x_train_val = np.concatenate((x_train, x_val))
y_train_val = np.concatenate((y_train['class'], y_val['class']))

# create model and train
clf = DecisionTreeClassifier(criterion = 'entropy', min_samples_split = 5)
clf.fit(x_train_val, y_train_val)

# make predictions
y_pred = clf.predict(x_test)
y_pred_train = clf.predict(x_train_val)

# print results
print('criterion: {}'.format('entropy'))
print('min_samples_split: {}'.format(5))
print ('\tTrain + Validation accuracy = ' + str(np.sum(y_pred_train == np.array(y_train_val))*1.0/len(y_train_val)))
print ('\tTest accuracy = ' + str(np.sum(y_pred == np.array(y_test['class']))*1.0/len(y_test)))
```

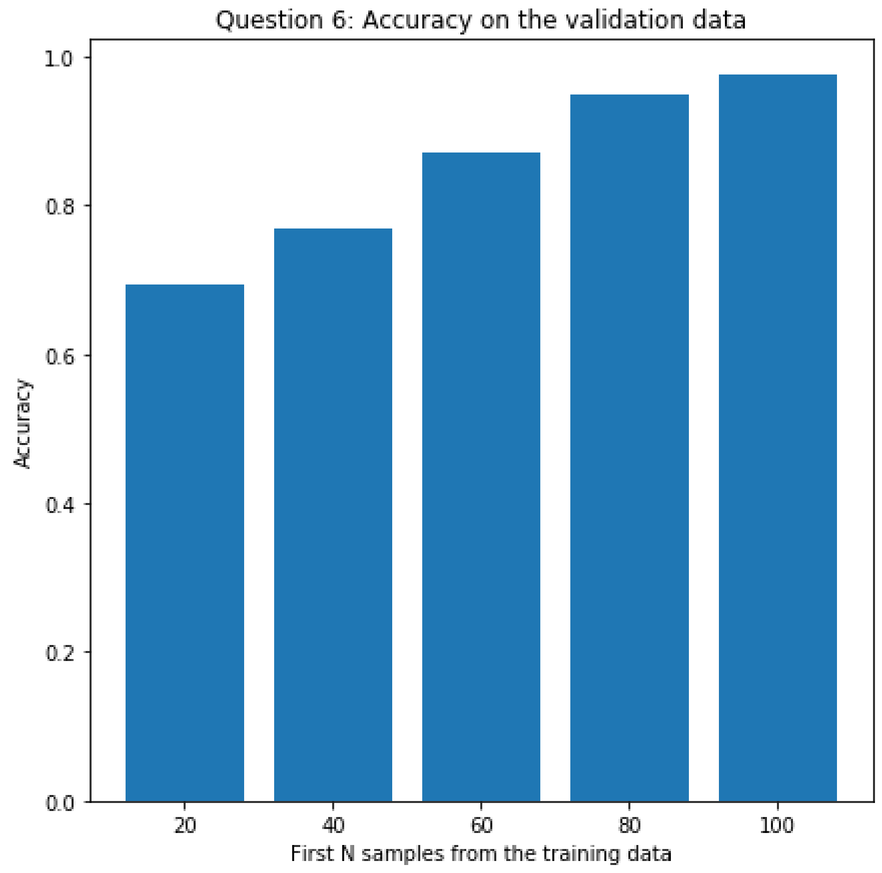
```
criterion: entropy
min_samples_split: 5
        Train + Validation accuracy = 1.0
        Test accuracy = 0.8205128205128205
```

Question 6: Use the parameters selected above (Q4 and Q5) to train Decision Tree model using the first 20, 40, 60, 80 and 100 samples from train data. Keep the validation set unchanged during this analysis. Report and plot the accuracies on the validation data. (2 marks)

```
In [169]: # train Decision Tree model using the first 20, 40, 60, 80 and 100 samples N from train data
accuracy_val_list = []
N_samples = [20,40,60,80,100]
for N in N_samples:

    # create model and train
    clf = DecisionTreeClassifier(criterion = 'entropy', min_samples_split = 5)
    clf.fit(x_train[:N], y_train[:N])

    # make predictions
    y_pred = clf.predict(x_val)
    y_pred_train = clf.predict(x_train[:N])

    # print results
    print('criterion: {}'.format('entropy'))
    print('min_samples_split: {}'.format(5))
    print('First {} samples from the train data'.format(N))
    print ('\tTrain accuracy = ' + str(np.sum(y_pred_train == np.array(y_train['class'][:N]))*1.0/len(y_train)))
    accuracy_val = np.sum(y_pred == np.array(y_val['class']))*1.0/len(y_val)
    print ('\tValidation accuracy = ' + str(accuracy_val))
    accuracy_val_list.append(accuracy_val)

# plot the accuracies on the validation data
plt.figure(figsize = (7,7))

plt.bar([str(i) for i in N_samples], accuracy_val_list)

plt.title('Question 6: Accuracy on the validation data')
plt.xlabel('First N samples from the training data')
plt.ylabel('Accuracy')

plt.show()
```

```
criterion: entropy
min_samples_split: 5
First 20 samples from the train data
        Train accuracy = 0.2
        Validation accuracy = 0.6923076923076923
criterion: entropy
min_samples_split: 5
First 40 samples from the train data
        Train accuracy = 0.4
        Validation accuracy = 0.7692307692307693
criterion: entropy
```

```
min_samples_split: 5
First 60 samples from the train data
        Train accuracy = 0.6
        Validation accuracy = 0.8717948717948718
criterion: entropy
min_samples_split: 5
First 80 samples from the train data
        Train accuracy = 0.79
        Validation accuracy = 0.9487179487179487
criterion: entropy
min_samples_split: 5
First 100 samples from the train data
        Train accuracy = 1.0
        Validation accuracy = 0.9743589743589743
```



```
In [ ]:
```