2-Adaboost

In [1]:

```
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from random import sample
import math
import random
from sklearn import tree
from sklearn.metrics import confusion_matrix
pd.set_option('display.max_rows', None)
```

In [2]:

```
class adaboost:
    def __init__(self):
        pass

    def sample(self, N, p):
        random_sample = np.zeros(N)
        p_estimate = np.zeros(len(p))
        p_cdf = np.cumsum(p)
        counts = np.zeros(len(p))

        for n in range(N):
            # generate a random number on [0,1]
            x = np.random.rand()
            random_sample[n] = np.where(((p_cdf > x)*1.0) == 1.)[0][0]
            counts[int(random_sample[n])] += 1

        p_estimate = counts/counts.sum()
        return random_sample, p_estimate

    def weakLearn(self, D, dataSet, DT=None):
        # DT model with depth one
        clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100
        #random distribution for the samples
        random.seed(10)
        rDataSet = dataSet.sample(len(D), replace = True, weights = D)

        X_train = rDataSet.iloc[:,0:4]
        y_train = rDataSet.iloc[:,4]

        #fitting the DT model
        stump = clf_gini.fit(X_train, y_train)

        return stump


    def fit(self, X, y, T):
        """
        X: feature vector
        y: labels
        T: number of iterations
        """
        self.trainingData = pd.DataFrame(X)
        self.trainingData['Label'] = y
        self.stumps = []
        self.alphas = []

        #Initially assign same weights to each records in the dataset
        self.trainingData['weights'] = 1/(self.trainingData.shape[0])

        for i in range(T):
            print("iteration {} ...".format(i+1))
            #create weak classifier
            stump = self.weakLearn(self.trainingData['weights'], self.trainingDa

            #append stumps
            self.stumps.append(stump)

            #make a prediction with the weak model
            y_pred = stump.predict(self.trainingData.iloc[:,0:4])


            #save the prediction
            self.trainingData['pred'] = y_pred

            #find the misclassified samples
            self.trainingData['misclassified'] = \
                                    np.where(self.trainingData['Label'] == self.trai


            #calulating the error
            e = sum(self.trainingData['misclassified'] * self.trainingData['weig

            #calculation of alpha
            alpha = 0.5*math.log((1-e)/e)

            self.alphas.append(alpha)

            #update weights
            new_weights = self.trainingData['weights']*np.exp(-1*alpha*self.trai
                                    *self.trainingData['pred'])

            #normalized weight
            z = sum(new_weights)
            normalized_weights = new_weights/z


            #add the new weights
            self.trainingData['weights'] = normalized_weights


    def predict(self, X):
        """
        Make prediction using the fitted model
        """
        stump_preds = np.array([stump.predict(X) for stump in self.stumps])
        return np.sign(np.dot(np.asarray(self.alphas), stump_preds))
```

Data preparation

In [3]:

```
blood = pd.read_csv("blood.csv", header=None)
blood.iloc[:,-1:] = blood.iloc[:,-1:].replace(to_replace = [1,0], value=[1,-1])
```

In [4]:

```
X = blood.iloc[:,0:4].values
X.shape
```

Out[4]:

```
(748, 4)
```

In [5]:

```
y = blood.iloc[:,4].values
y.shape
```

Out[5]:

```
(748,)
```

Training and predicting

In [6]:

```
obj = adaboost()
```

In [7]:

```
obj.fit(X, y, 20)
```

```
iteration 1 ...
iteration 2 ...
iteration 3 ...
iteration 4 ...
iteration 5 ...
iteration 6 ...
iteration 7 ...
iteration 8 ...
iteration 9 ...
iteration 10 ...
iteration 11 ...
iteration 12 ...
iteration 13 ...
iteration 14 ...
iteration 15 ...
iteration 16 ...
iteration 17 ...
iteration 18 ...
iteration 19 ...
iteration 20 ...
```

In [8]:

```
stump_preds = obj.predict(X)
```

Comparing accuracy to sklearn’simplementation

In [9]:

```
#Using the confusion matrix for evaluating the accuracy
c=confusion_matrix(y, stump_preds)
c
```

Out[9]:

```
array([[543,  27],
       [118,  60]])
```

In [10]:

```
#Accuracy
(c[0,0]+c[1,1])/np.sum(c)*100
```

Out[10]:

```
80.61497326203208
```

In [11]:

```
X_train = X
y_train = y
```

In [12]:

```
clf = AdaBoostClassifier(n_estimators=20, random_state=0)
clf.fit(X_train, y_train)
```

Out[12]:

```
AdaBoostClassifier(n_estimators=20, random_state=0)
```

In [13]:

```
clf.score(X_train, y_train)
```

Out[13]:

```
0.8074866310160428
```

In [ ]: