Programming for Artificial Intelligence (Python)
Homework 6 & 7
Due: May 8 before class

This homework focuses on implementing the Adaboost algorithm from scratch.

0.Necessary libraries

For this homework, you will need several libraries:

1. `numpy`
2. `sklearn`
3. `matplotlib`

You probably have installed `numpy`, and `functools` is a built-in package. So you will only need to install `sklearn` and `matplotlib`. To do that, go into your Python environment in the conda prompt (from here:) and activate your environment. Then type

```
conda install -y scikit-learn
conda install -y matplotlib
```

After installation, load the packages

```python
from sklearn import datasets
import numpy as np
from functools import partial
from sklearn.tree import plot_tree
```

Make sure you can load the packages successfully.

1.The Iris dataset

Iris is a popular data set for classification tasks. This data is embedded in `sklearn` so we don't need to type in values by ourselves. To load the data,

```
iris = datasets.load_iris()
Xall = iris.data
conames = iris.feature_names
yall = iris.target
iris.target_names
```

How many classes are there in the Iris data? How many covariates does it have?

2.Preprocessing data

Since we only talked about binary classification in class, let's first take a subset of the Iris data. Meanwhile, we will only use two covariates so that we can visulize the data.

```
index_want = yall < 2
X = Xall[index_want, 0:2]
y = (yall[index_want] - 0.5) * 2
print(X.shape, y.shape)
```

Interpret the above code.

3.Visulization

To plot the data, there is a package called `plotiris`. But unlike other pacakges, this package is provided by your Professor, so you cannot download it from the Internet. To load this package, download the `plotiris.py` file from SPOC and put it into the folder where your homeword notebook is stored. Then run

```
import plotiris
plotiris.plotiris(y, X)
```

Draw the scatter plot.

4.Initialization

Now we are ready to implement the Adaboost algorithm. The first step is to initialize the point weights $w_{t,i}$ (these are the weights of each observation), where t means the weights to be used in the creation of the t-th tree and i means the i-th observation/sample/data/exmaple. **Note: we do not have α yet because α will depend on the first tree.**

Complete the following function so that $w_{t,i} = 1/N$, where N is the number of observations in our data.

```python
# initialization
def initAda(y):
  pass
w = initAda(y)
```

This is all we need before we start the first tree!

5.The first tree

From now on, you need the following algorithm (you already initialized $w_{t,i}$):

1. Find the t-th tree $f_t(\boldsymbol{x})$ by
$$\min \sum_{y_i \neq f_t(\boldsymbol{x}_i)} w_{t,i} \tag{1}$$
2. Compute error of the t-th tree as
$$E_t = \sum_{y_i \neq f_t(\boldsymbol{x}_i)} w_{t,i} \tag{2}$$
3. ompute tree weight:
$$\alpha_t = \frac{1}{2} \log \left(\frac{1 - E_t}{E_t} \right) \tag{3}$$
4. Ensemble:
$$F_t(\boldsymbol{x}) = F_{t-1}(\boldsymbol{x}) + \alpha_t f_t(\boldsymbol{x}) \tag{4}$$
5. Update data weights:
$$w_{t+1,i} = w_{t,i} e^{-y_i \alpha_t f_t(\boldsymbol{x}_i)} \tag{5}$$

Recall that if we want to make a tree with only one cut (which is also known as a stump), we need to write a nested loop. For the outer loop, we will go over the covariates and for the inner loop, we will search for where to cut. In general, the nested loop looks like:

```
for i in range(number_of_covariates):
  for cut in possible_cuts_of_ith_covariate:
    pass
```

About the inner loop, we do not need to go over all real values $\mathbb{R}$. Why? Suppose our x_1 has two values: 0.3 and 0.7. Does it make any difference when we cut at $x_1 = 0.4$ and at $x_1 = 0.6$?

According to equation (**??**), complete the code below (by replacing the ???) to grow a tree

```
# find the stump
def stump(y, X, w):
    best_covar = None
    best_cut = None
    best_error = np.inf
    yhat = None

    for covar in range(X.shape[1]):
        cuts = ???
        for cut in cuts:
            ytemp = np.ones(len(y))
            # we are not sure which value to assign
            if ??? < 0:
                ??? = -1
            else:
                ??? = 1
            error = np.sum(w[y ≠ ytemp])
            # when do we update information?
            if error < best_error:
                ???
    print(f"best: {conames[best_covar]} at {best_cut}"
  )
    return best_covar, best_cut, best_error, yhat
```

6. Compute the error

According to equation (**??**), complete the following function to compute the total error of the t-th tree.

```
# compute error
def compute_error(y, yhat, w):
    pass
```

Suppose you call the `stump` function

```
cut = stump(y, X, w)
```

How would you use the `compute_error` function to find the total error?

```
E = ???
```

7. Weight of a tree

Now you have grown the first tree and computed its total error. Note that only after this can you find the weight of the tree.

According to equation (**??**), complete the code below to help you find the weight of a tree.

```
# compute stump weight:
def getalpha(E):
    pass
a = getalpha(E)
```

Why cannot we find the weight before the tree?

8. Update point weights

We've got all we need. But let's skip equation (**??**) for now. According to equation (**??**), how would you complete the following code to update w.

```
# update weights
def update_point_weight(???):
    pass
w = update_point_weight(???)
```

9.Putting 'em all together

Did you notice where you started? You started from w. Did you notice where you ended with? Also w! Great. That means you have finished one iteration of a loop!

Let's now combine all functions into a loop:

```
ff = {}
alpha = []
T = 3
w = init(y)
for (t in range(T)):
  cut = ??? # equation 1
  E = ??? # equation 2
  a = getalpha(E) # equation 3
  alpha.append(a)
  ff[t] = cut[0] # equation 4
  w = ??? #equation 5
```

What are the final estimations/predictions $\hat{y}_i$?

```
yhat_naive = ???
```

10.Improve the tree

In the above steps, you can step into them by going over the loop manually. Then you can plot the trees using

```
plotiris(y, X, w, addvline=cut[1])
```

of

```
plotiris(y, X, w, addhline=cut[1])
```

When you do it four times, you will notice that the tree cannot improve. This is because that our algorithm is a simplified version of Adaboost. One way to improve our algorithm is to change our one `stump` function to the tree function of `sklearn`:

```
from sklearn.tree import DecisionTreeClassifier
def step1(y, X, w):
    G_m = DecisionTreeClassifier(max_depth = 1)     # Stump: Two terminal-node classification tree
    G_m.fit(X, y, sample_weight = w)
    y_pred = G_m.predict(X)
    return y_pred, G_m
```

Make necessary changes to your previous functions to improve our Adaboost algorithm.

```
yhat_improve = ???
```

11.Using others' wheel

We have gone so much to write our own Adaboost. This helped us to understand this algorithm. However, `sklearn` have alreday done so for us. Read this page https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html and complete the following to get a prediction with three trees. Compare it with our prediction in question 10.

```
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(n_estimators=100, algorithm="SAMME", random_state=0)
clf.fit(X, y)
yhat_sklearn = ???
compare = yhat_improve * yhat_sklearn
```

12.Plot a tree (Optional)

If you are curious about what each one tree looks like, you can do:

```
plot_tree(cut[1])
```