```
from libsvm.svmutil import *
import numpy as np

train_y, train_x = svm_read_problem('train1.txt')
test_y, test_x = svm_read_problem('test1.txt')

train_x = [list(train_x[i].values()) for i in range(len(train_x)) if train_y
[i] in [11, 26]]
test_x = [list(test_x[i].values()) for i in range(len(test_x)) if test_y[i] in
[11, 26]]
train_y = [1 if x == 11 else -1 if x == 26 else None for x in train_y if x in
[11, 26]]
test_y = [1 if x == 11 else -1 if x == 26 else None for x in test_y if x in [1
1, 26]]


class DecisionStump:
    def __init__(self, dimension, threshold, direction):
        self.dimension = dimension
        self.threshold = threshold
        self.direction = direction

    def predict(self, X):
        return -self.direction * np.sign(X[:, self.dimension] - self.threshol
d)

class AdaBoost:
    def __init__(self, n_estimators):
        self.n_estimators = n_estimators
        self.estimators = []
        self.alphas = []
        self.allerr = []
        self.ccc = 0
    def fit(self, X, y):
        n_samples = X.shape[0]
        weights = np.full(n_samples, 1 / n_samples)

        for _ in range(self.n_estimators):
            #print(_)
            best_stump = self._find_best_stump(X, y, weights)
            #self.ccc=1
            error = self._compute_error(best_stump, X, y, weights)
            alpha = 0.5 * np.log((1 - error) / error)
            #print(error)
            self.estimators.append(best_stump)
            self.alphas.append(alpha)

            predictions = best_stump.predict(X)
            #print(np.array(predictions==y).astype(int))
            weights *= np.exp(-alpha * y * predictions)
```

```
#             print(len(weights))
            #print(sum(weights))

            weights /= np.sum(weights)
#             print(len(weights))
#             print(sum(weights))
            eps = np.sum(predictions != y)/len(y)
            #print()
            self.allerr.append(eps)

    def predict(self, X):
        predictions = np.zeros(X.shape[0])
        for stump, alpha in zip(self.estimators, self.alphas):
            predictions += alpha * stump.predict(X)
        return np.sign(predictions)

    def _find_best_stump(self, X, y, weights):
        n_samples, n_features = X.shape
        best_stump = None
        min_error = float('inf')
        #cnt=0
        for dimension in range(n_features):
            sorted_indices = np.argsort(X[:, dimension])
            #print(len(X[sorted_indices, dimension]))
            s_X = X[sorted_indices, dimension].copy()
            #unique_values = np.unique(X[:, dimension])
            #print('u', s_X)
            thresholds = np.unique((s_X[:-1] + s_X[1:]) / 2)
            #print('aaa', thresholds)
            a = np.array(-100000000)
            thresholds = np.append(a,thresholds)
            #print(thresholds)
            for threshold in thresholds:
                for direction in [-1, 1]:
                    #cnt+=1
                    stump = DecisionStump(dimension, threshold, direction)
                    error = self._compute_error(stump, X, y, weights)
                    #self.allerr.append(error)
                    if error < min_error:
                        min_error = error
                        best_stump = stump
        #print(min_error)
        #self.allerr.append(min_error)
        #print(self.ccc, best_stump.dimension, best_stump.threshold,best_stum
p.direction)
        self.ccc+=1
        return best_stump

    def _compute_error(self, stump, X, y, weights):
        predictions = stump.predict(X)
        #if self.ccc==1:
            #print(sum(predictions))
```

```
        incorrect = predictions != y
        return np.sum(weights[incorrect])


X = np.array(train_x)
y = np.array(train_y)

adaboost = AdaBoost(n_estimators=1000)
adaboost.fit(X, y)

test_X = np.array(test_x)
test_y = np.array(test_y)

print("min Ein(g): ",min(adaboost.allerr))
print("max Ein(g): ",max(adaboost.allerr))

pred_Ein_G = adaboost.predict(X)
print(np.mean(y != pred_Ein_G))

pred_Eout_G = adaboost.predict(test_X)
print(np.mean(test_y != pred_Eout_G))

# min Ein(g):  0.09846547314578005
# max Ein(g):  0.571611253196931
# 0.0
# 0.002793296089385475
```