AdaBoost - Lab

September 16, 2021

Modify the AdaBoost scratch code in our lecture such that: - Notice that if err = 0, then α will be undefined, thus attempt to fix this by adding some very small value to the lower term - Notice that sklearn version of AdaBoost has a parameter learning_rate. This is in fact the $\frac{1}{2}$ in front of the α calculation. Attempt to change this $\frac{1}{2}$ into a parameter called eta, and try different values of it and see whether accuracy is improved. Note that sklearn default this value to 1. - Observe that we are actually using sklearn DecisionTreeClassifier. If we take a look at it closely, it is actually using weighted gini index, instead of weighted errors that we learn above. Attempt to write your own class of class Stump that actually uses weighted errors, instead of weighted gini index. To check whether your stump really works, it should give you still relatively the same accuracy. In addition, if you do not change y to -1, it will result in very bad accuracy. Unlike sklearn version of DecisionTree, it will STILL work even y is not change to -1 since it uses gini index - Put everything into a class

[1]:
```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=500, random_state=1)
y = np.where(y==0,-1,1)  #change our y to be -1 if it is 0, otherwise 1

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
```

[3]:
```
class AdaBoost():
    def __init__(self, S=5, eta=0.5):
        self.S = S
        self.eta = eta
        self.stump_params = {'max_depth': 1, 'max_leaf_nodes': 2}
        self.models = models = [DecisionTreeClassifier(**self.stump_params) for␣
 ↪_ in range(S)]

    def fit(self, X, y):
        m = X_train.shape[0]

        #initially, we set our weight to 1/m
```

```
        self.W = np.full(m, 1/m)

        #keep collection of a_j
        self.a_js = np.zeros(self.S)

        for j, model in enumerate(self.models):

            #train weak learner
            model.fit(X_train, y_train, sample_weight = self.W)

            #compute the errors
            yhat = model.predict(X_train)
            err = self.W[(yhat != y_train)].sum()

            #compute the predictor weight a_j
            #if predictor is doing well, a_j will be big
            peb = 1e-5 # prevent divide by zero
            a_j = np.log ((1 - err) / (err + peb))
            self.a_js[j] = a_j

            #update sample weight; divide sum of W to normalize
            self.W = (self.W * np.exp(-a_j * y_train * yhat))
            self.W = self.W / sum (self.W)

    def predict(self, X_test):
        #make weighted predictions
        Hx = 0
        for i, model in enumerate(self.models):
            yhat = model.predict(X_test)
            Hx += self.a_js[i] * yhat

        yhat = np.sign(Hx)
        return yhat
```

[4]:
```
class Stump():
    def __init__(self):
        # Determines whether threshold should be evaluated as < or >
        self.polarity = 1
        self.feature_index = None
        self.threshold = None
        # Voting power of the stump
        self.alpha = None
```

[5]:
```
class AdaBoostStump():
    def __init__(self, S=5, eta=0.5):
        self.S = S
        self.eta = eta
```

```
        self.clfs = []

    def fit(self, X, y):
        m, n = X.shape

        #initially, we set our weight to 1/m
        self.W = np.full(m, 1/m)

        #keep collection of a_j
        self.a_js = np.zeros(self.S)

        for _ in range(self.S):
            clf = Stump()

            #set initially minimum error to infinity
            #so at least the first stump is identified
            min_err = np.inf

            #previously we don't need to do this
            #since sklearn learn does it
            #but now we have to loop all features, all threshold
            #and all polarity to find the minimum weighted errors
            for feature in range(n):
                feature_vals = np.sort(np.unique(X[:, feature]))
                thresholds = (feature_vals[:-1] + feature_vals[1:])/2
                for threshold in thresholds:
                    for polarity in [1, -1]:
                        yhat = np.ones(len(y)) #set all to 1
                        yhat[polarity * X[:, feature] < polarity * threshold] =␣
↪-1  #polarity=1 rule
                        err = self.W[(yhat != y)].sum()

                        #save the best stump
                        if err < min_err:
                            clf.polarity = polarity
                            clf.threshold = threshold
                            clf.feature_index = feature
                            min_err = err

            #compute the predictor weight a_j
            #if predictor is doing well, a_j will be big
            peb = 1e-5 # prevent divide by zero
            a_j = np.log ((1 - err) / (err + peb))
            clf.alpha = a_j

            #update sample weight; divide sum of W to normalize
            self.W = (self.W * np.exp(-clf.alpha * y_train * yhat))
```

```
            self.W = self.W / sum (self.W)
            self.clfs.append(clf)

    def predict(self, X):
        m, n = X.shape
        yhat = np.zeros(m)
        for clf in self.clfs:
            pred = np.ones(m) #set all to 1
            pred[clf.polarity * X[:, clf.feature_index] < clf.polarity * clf.
↪threshold] = -1  #polarity=1 rule
            yhat += clf.alpha * pred

        return np.sign(yhat)
```

[6]:
```
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

model = AdaBoost(S=20, eta=0.5)
model.fit(X_train, y_train)
yhat = model.predict(X_test)
print(classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

          -1       0.92      0.99      0.95        79
           1       0.98      0.90      0.94        71

    accuracy                           0.95       150
   macro avg       0.95      0.94      0.95       150
weighted avg       0.95      0.95      0.95       150
```

[7]:
```
model = AdaBoostStump(S=20, eta=0.5)
model.fit(X_train, y_train)
yhat = model.predict(X_test)
print(classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

          -1       0.94      0.95      0.94        79
           1       0.94      0.93      0.94        71

    accuracy                           0.94       150
   macro avg       0.94      0.94      0.94       150
weighted avg       0.94      0.94      0.94       150
```

[8]:
```
from sklearn.ensemble import AdaBoostClassifier

#SAMME.R - a variant of SAMME which relies on class probabilities
#rather than predictions and generally performs better
ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    learning_rate=0.07, random_state=42)
ada_clf.fit(X_train, y_train)
y_pred = ada_clf.predict(X_test)
print("Ada score: ", accuracy_score(y_test, y_pred))
```

```
Ada score:  0.9733333333333334
```

[ ]:
```

```