AdaBoost - Lab

September 20, 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]:

```python
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.metrics import classification_report

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)
```

[2]:

```python
class DecisionStump():
    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
```

[3]:

```python
class AdaBoost():
    def __init__(self, n_estimators=5, eta=0.5):
        self.n_estimators = n_estimators
        self.eta = eta
    def fit(self, X, y):
        m, n = X.shape

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

        self.clfs = []

        # create models
        for _ in range(self.n_estimators):
            clf = DecisionStump()
            min_err = np.inf

            # loop through all fetures
            for feature in range(n):
                feature_vals = np.sort(np.unique(X[:, feature])) # sort current␣
↪feature
                thresholds = (feature_vals[:-1] + feature_vals[1:])/2 # get the␣
↪thresholds eg. [2,4,6]: (2,4 + 4,6) / 2 = 6/2,10/2 = 3,5

                # loop through each threshold
                for threshold in thresholds:

                    for polarity in [1, -1]:
                        yhat = np.ones(len(y)) # set all to 1

                        # when polarity = 1 if feature < threshold, then -1,␣
↪else 1
                        # when polarity = -1 if feature < threshold, then 1,␣
↪else -1
                        yhat[polarity * X[:, feature] < polarity * threshold] =␣
↪-1


                        err = W[(yhat != y)].sum()

                        if err < min_err:
                            clf.polarity = polarity
                            clf.threshold = threshold
                            clf.feature_index = feature
                            min_err = err
            eps = 1e-10
            clf.alpha = self.eta * (np.log((1 - err) / (err + eps)))
            W = W * np.exp(-clf.alpha * y * yhat)
            W = W / sum (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)
            pred[clf.polarity * X[:, clf.feature_index] < clf.polarity * clf.
 ↪threshold] = -1
            yhat += clf.alpha * pred
        return np.sign(yhat)
```

[4]:

```
model = AdaBoost(n_estimators=10)
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
```