Gradient Boosting - Lab

September 26, 2021

Modify the Gradient Boosting scratch code in our lecture such that: - Notice that we are still using max_depth = 1. Attempt to tweak min_samples_split, max_depth for the regression and see whether we can achieve better mse on our boston data - Notice that we only write scratch code for gradient boosting for regression, add some code so that it also works for binary classification. Load the breast cancer data from sklearn and see that it works. - Further change the code so that it works for multiclass classification. Load the digits data from sklearn and see that it works - Put everything into class

[1]:

```
from scipy.special import expit
from sklearn.tree import DecisionTreeRegressor
from sklearn.dummy import DummyRegressor
import numpy as np
```

[2]:

```
class GradientBoosting:
    def __init__(self, S=5, learning_rate=1, max_depth = 1, min_samples_split =␣
 ↪2, regression=True, tol=1e-4):
        self.S = S
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.regression=regression

        #initialize regression trees
        tree_params = {'max_depth': self.max_depth,
                       'min_samples_split': self.min_samples_split}
        self.models = [DecisionTreeRegressor(**tree_params) for _ in range(S)]
        first_model = DummyRegressor(strategy='mean')
        self.models.insert(0, first_model)

    def tranform_to_n_dimensions(self, y_train, y):
        y_train_encoded = np.zeros((y_train.shape[0], len(set(y))))
        for each_class in range(len(set(y))):
            cond = y_train==each_class
            y_train_encoded[np.where(cond), each_class] = 1
        return y_train_encoded

    def grad(self, y, h):
        return y - h
    def fit(self, X, y):
        #using DummyRegressor is a good technique for starting model
        self.models[0].fit(X, y)

        #fit the estimators
        for i in range(self.S):
            #predict using all the weak learners we trained up to
            #this point
            y_pred = self.predict(X, self.models[:i+1], with_argmax=False)

            #errors will be the total errors maded by models_trained
            residual = self.grad(y, y_pred)

            #fit the next model with residual
            self.models[i+1].fit(X, residual)

    def predict(self, X, models=None, with_argmax=True):
        if models is None:
            models = self.models
        learning_rate = 0.1  ##hard code for now
        f0 = models[0].predict(X)  #first use the dummy model
        boosting = sum(self.learning_rate * model.predict(X) for model in␣
 ↪models[1:])
        yhat = f0 + boosting

        if not self.regression:
            #turn into probability using softmax
            yhat = np.exp(yhat) / np.sum(np.exp(yhat), axis=1, keepdims=True)
            if with_argmax:
                yhat = np.argmax(yhat, axis=1)
        return yhat
```

[3]:

```
# Regression

from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor

X, y = load_boston(return_X_y=True)

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

model = GradientBoosting(S=200, learning_rate=0.1, max_depth = 3,
                 min_samples_split = 2,
                 regression=True, tol=1e-4)
model.fit(X_train, y_train)
yhat = model.predict(X_test)

print("Our MSE: ", mean_squared_error(y_test, yhat))
```

```
Our MSE:  7.868386054241114
```

[4]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import accuracy_score

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,␣
 ↪random_state=42)

model = GradientBoosting(S=200, learning_rate=0.1, max_depth = 3,
                 min_samples_split = 2,
                 regression=False)

# seperate y into 2 classes
# thus, dimension of y should be (m, 2) binary classification
y_train = model.tranform_to_n_dimensions(y_train, y)

model.fit(X_train, y_train)
yhat = model.predict(X_test)

print(y_test)
print(yhat)
print("Our accuracy: ", accuracy_score(y_test, yhat))
```

```
[1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0
 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0
 1 1 1 0 1 1 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0
 1 1 0 1 0 1 1 1 0 1 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 1 0 1 0 1 1 0 1 0 0
 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1]
[1 0 0 1 1 0 0 0 0 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0
 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0
 1 1 1 0 1 1 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 0 1 0
 1 1 0 1 0 1 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 0 1 1 0 1 0 1 1 0 1 0 0
 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1]
Our accuracy:  0.9649122807017544
```

[5]:
```
# Multiclass classification
from sklearn.datasets import load_digits
```

```
X, y = load_digits(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,␣
 ↪random_state=42)

model = GradientBoosting(S=200, learning_rate=0.1, max_depth = 3,
                 min_samples_split = 2,
                 regression=False)

# seperate y into 9 classes
# thus, dimension of y should be (m, 9) binary classification
y_train = model.tranform_to_n_dimensions(y_train, y)

model.fit(X_train, y_train)
yhat = model.predict(X_test)

# #print metrics
print("Our accuracy: ", accuracy_score(y_test, yhat))
```

```
Our accuracy:  0.9314814814814815
```

[ ]:

```
```