Gradient Boosting - Lab

September 25, 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 sklearn.tree import DecisionTreeRegressor
from sklearn.dummy import DummyRegressor
from sklearn.model_selection import train_test_split
import numpy as np
```

[2]:
```
class GradientBoosting:
    def __init__(self, n_estimators=5, learning_rate=1, max_depth = 1,␣
↪min_samples_split = 2, regression=True):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        tree_params = {'max_depth': self.max_depth,
                'min_samples_split': self.min_samples_split}
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.regression = regression
        self.models = [DecisionTreeRegressor(**tree_params) for _ in␣
↪range(n_estimators)]
        first_model = DummyRegressor(strategy='mean')
        self.models.insert(0, first_model)

    def grad(self, y, h):
        return y - h

    def fit(self, X, y):
        self.models[0].fit(X, y)

        for i in range(self.n_estimators):
            y_pred =  self.predict(X, self.models[:i+1], with_argmax=False)

            residual = self.grad(y, y_pred)
```

```
            self.models[i+1].fit(X, residual)


    def predict(self, X, models=None, with_argmax=True):
        if models is None:
            models = self.models
        f0 = models[0].predict(X)
        boosting = sum(self.learning_rate * model.predict(X) for model in␣
↪models[1:])
        y_pred = f0 + boosting

        # if the task is classification, apply softmax function to the␣
↪predicted value
        if not self.regression:
            y_pred = np.exp(y_pred) / np.sum(np.exp(y_pred), axis=1,␣
↪keepdims=True)
            if with_argmax:
                y_pred = np.argmax(y_pred, axis=1)
        return y_pred
```

[3]:
```
# regression

from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error
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)

n_estimators = 200
learning_rate = 0.1

# setting max_depth to 3, min_samples_split to 2
max_depth = 3
min_samples_split = 2

model = GradientBoosting(n_estimators=n_estimators,
                         learning_rate=learning_rate,
                         max_depth = max_depth,
                         min_samples_split = min_samples_split,)
model.fit(X_train, y_train)
yhat = model.predict(X_test)

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

```
# sklearn
sklearn_model = GradientBoostingRegressor(n_estimators=n_estimators,
                                          learning_rate=learning_rate,
                                          max_depth=max_depth,
                                          min_samples_split=min_samples_split,
                                          loss='ls')
yhat_sk = sklearn_model.fit(X_train, y_train).predict(X_test)
print("Sklearn MSE: ", mean_squared_error(y_test, yhat_sk))
```

```
Our MSE:  7.896061062860724
Sklearn MSE:  7.996273842366492
```

[4]:
```
# binary classification

from sklearn.datasets import load_breast_cancer
from sklearn.metrics import accuracy_score
from sklearn.ensemble import GradientBoostingClassifier

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)
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

model = GradientBoosting(n_estimators=200, learning_rate=0.1, max_depth = 3,
                 min_samples_split = 2,
                 regression=False)
model.fit(X_train, y_train_encoded)
yhat = model.predict(X_test)


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

# sklearn
sklearn_model = GradientBoostingClassifier(
    n_estimators=n_estimators,
    learning_rate = 0.1,
    max_depth=1
)

yhat_sk = sklearn_model.fit(X_train, y_train).predict(X_test)
print("Sklearn accuracy: ", accuracy_score(y_test, yhat_sk))
```

```
Our accuracy:  0.9649122807017544
Sklearn accuracy:  0.9649122807017544
```

[5]:
```
# multi-class classification

from sklearn.datasets import load_digits
from sklearn.metrics import accuracy_score
from sklearn.ensemble import GradientBoostingClassifier

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)
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

model = GradientBoosting(n_estimators=200, learning_rate=0.1, max_depth = 3,
                 min_samples_split = 2,
                 regression=False)
model.fit(X_train, y_train_encoded)
yhat = model.predict(X_test)

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



# sklearn
sklearn_model = GradientBoostingClassifier(
    n_estimators=n_estimators,
    learning_rate = 0.1,
    max_depth=1
)

yhat_sk = sklearn_model.fit(X_train, y_train).predict(X_test)
print("Sklearn accuracy: ", accuracy_score(y_test, yhat_sk))
```

```
Our accuracy:  0.9314814814814815
Sklearn accuracy:  0.9481481481481482
```

[ ]:
```

```