# 01 - Supervised Learning - Classification - Gradient Boosting(Solution)_st122097_thantham

September 22, 2021

# 1 Programming for Data Science and Artificial Intelligence

## 1.1 Classification - Gradient Boosting

## 1.2 Name: Thantham Khamyai

## 1.3 Student ID: 122097

### 1.3.1 ===Task===

Modify the above scratch code 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

### 1.3.2 TASK 4 Making Gradient Boosting Class

[1]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.dummy import DummyRegressor
import numpy as np

class GradientBoosting:

    def __init__ (self, estimator = None , model_params={'max_depth':3,␣
↪'min_samples_split':2},
                  n_estimators=100, is_regression = True,
                  learning_rate=0.1):

        if estimator == None: # Just leave this, in case future we would put␣
↪other regressor model
            estimator = DecisionTreeRegressor

        self.n_estimators = n_estimators # get number of estimators
```

```
        # init model list
        self.models =␣
↪[DummyRegressor(strategy='mean')]+[estimator(**model_params) for _ in␣
↪range(self.n_estimators)]

        self.learning_rate = learning_rate # alpha
        self.is_regression = is_regression # to check if it is regression or␣
↪classification


    def fit(self, X, y):

        self.models[0].fit(X, y) # fit first dummy regressor

        current_model = 1 # init first boosting idx

        while current_model != len(self.models): # while not all boosters ->␣
↪keep fit them

            # predict model i and keep not result output class
            h_x = self.predict(X, self.models[:current_model],␣
↪argmax_output=False)

            residual = self.residual(y, h_x) # find residual (gradient)

            self.models[current_model].fit(X, residual) # try to fit X to␣
↪residual prediction

            current_model += 1 # next model



    def residual(self, y, h_x):
        return y - h_x    # simple y - h(x)


    def predict(self, X, estimators=None, argmax_output=True):

        if estimators == None: # if predict was used from outside
            estimators = self.models # define all models to be predictors

        # predict H(x) by addition of first dummy with other regressor␣
↪predictions * alpha
        H_X = self.models[0].predict(X) + sum(self.learning_rate * self.
↪models[i].predict(X) for i in range(1, len(estimators)))
```

```
        if not self.is_regression: # if this boosting is classificaiton

            H_X = np.exp(H_X) / np.sum(np.exp(H_X), axis=1, keepdims=True) #␣
↪implement softmax (works with binary and multiclass)

            if argmax_output: # if predict method was used from outside

                H_X = np.argmax(H_X, axis=1) # return class of predicted

        return H_X
```

[2]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import␣
↪GradientBoostingRegressor,GradientBoostingClassifier

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

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

from sklearn.datasets import load_digits
from sklearn.metrics import classification_report
```

### 1.3.3 TASK 1 Implementing Regression Gradient Boosting on Boston Data

[3]:
```
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=48)
```

After loading Boston, lets apply out gradient boosting model

[4]:
```
model_boston = GradientBoosting(n_estimators=200, learning_rate=0.1,␣
↪model_params={'max_depth':3, 'min_samples_split':2}, is_regression=True)
model_boston.fit(X_train, y_train)
ypred_boston = model_boston.predict(X_test)
print("Our model Mean square error: ", mean_squared_error(y_test, ypred_boston))
```

```
Our model Mean square error:  10.917610709239431
```

However, we have tried on max depth 1 or 3 before, lets change some to max depth 5 and min split to 4

[5]:
```
model_boston = GradientBoosting(n_estimators=200, learning_rate=0.1,␣
↪model_params={'max_depth':5, 'min_samples_split':4}, is_regression=True)
model_boston.fit(X_train, y_train)
```

```
ypred_boston = model_boston.predict(X_test)
print("Our model Mean square error: ", mean_squared_error(y_test, ypred_boston))
```

```
Our model Mean square error:  11.767735935158402
```

We found that there was some lower MSE after increasing max depth and min sample split. This tells us that there was some change if we try on different hyperparameter of regressors inside models.

However, to reach better result, we cannot exactly find what hyperparameters can do, we need **Cross-Validation** to automatically do that.

[6]:
```
sklearn_boston = GradientBoostingRegressor(n_estimators=200,learning_rate = 0.
 ↪1,max_depth=3, loss='ls')
sklearn_boston.fit(X_train, y_train)
ypred_sk_boston = sklearn_boston.predict(X_test)

print("Sklean Mean square error: ", mean_squared_error(y_test, ypred_sk_boston))
```

```
Sklean Mean square error:  12.35891581425541
```

### 1.3.4 TASK 2 Implementing Binary Classification Gradient Boosting on Breast Cancer Data

[7]:
```
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=48)
```

For The binary classification, we can directly implement softmax because softmax can do any binary or muiticlass classification.

To address this, we need to encode by onehot method for y_train

[8]:
```
def onehot(y):

    y_onehot = np.zeros((y.shape[0], len(set(y))))

    for class_i in range(len(set(y))): # to encode onehot -> loop each output

        y_at_class_i = y_train == class_i # get idx which y = class_i

        y_onehot[np.where(y_at_class_i), class_i] = 1 # change onehot row to be␣
 ↪1 at column class_i

    return y_onehot
```

Simply implement onehot function to y_train

[9]:
```
y_train_onehot = onehot(y_train)

print('Convert binary class to One hot', set(y_train))
print('y train as :', y_train_onehot[:10])
```

```
Convert binary class to One hot {0, 1}
y train as : [[1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [0. 1.]]
```

However, we don't need to convert y_test because predictions will be integer directly, and we can examine result via integer class

[10]:
```
model_cancer = GradientBoosting(n_estimators=200, learning_rate=0.1,␣
↪model_params={'max_depth':3}, is_regression=False)
model_cancer.fit(X_train, y_train_onehot)
yhat_cancer = model_cancer.predict(X_test)

# #print metrics
print("Our classification report:\n ", classification_report(y_test,␣
↪yhat_cancer))
```

```
Our classification report:
               precision    recall  f1-score   support

           0       0.87      0.95      0.91        76
           1       0.95      0.88      0.92        95

    accuracy                           0.91       171
   macro avg       0.91      0.92      0.91       171
weighted avg       0.92      0.91      0.91       171
```

[11]:
```
sklearn_cancer = GradientBoostingClassifier(n_estimators=200,learning_rate = 0.
↪1, max_depth=1)
sklearn_cancer.fit(X_train, y_train)
yhat_sk_cancer = sklearn_cancer.predict(X_test)
print("Sklearn classification report:\n ", classification_report(y_test,␣
↪yhat_sk_cancer))
```

```
Sklearn classification report:
```

```
              precision    recall  f1-score   support

           0       0.94      0.95      0.94        76
           1       0.96      0.95      0.95        95

    accuracy                           0.95       171
   macro avg       0.95      0.95      0.95       171
weighted avg       0.95      0.95      0.95       171
```

### 1.3.5 TASK 3 Implementing Multiclass Gradient Boosting on Digits Data

[12]:
```
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=48)

y_train_onehot = onehot(y_train)
print('Convert binary class to One hot', set(y_train))
print('y train as :', y_train_onehot[:10])
```

```
Convert binary class to One hot {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
y train as : [[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]]
```

One hot was performed the same as binary

[13]:
```
model_digit = GradientBoosting(n_estimators=200, learning_rate=0.1,␣
 ↪is_regression=False)
model_digit.fit(X_train, y_train_onehot)
yhat_digit = model_digit.predict(X_test)

# #print metrics
print("Our classification report:\n ", classification_report(y_test,␣
 ↪yhat_digit))
```

```
Our classification report:
               precision    recall  f1-score   support

           0       0.96      0.95      0.95        56
```

```
           1       0.97      0.91      0.94        69
           2       0.98      0.96      0.97        52
           3       0.98      0.94      0.96        47
           4       0.97      0.95      0.96        63
           5       0.88      0.93      0.90        40
           6       0.97      0.98      0.98        64
           7       0.94      0.98      0.96        52
           8       0.87      0.82      0.84        55
           9       0.80      0.93      0.86        42

    accuracy                           0.94       540
   macro avg       0.93      0.93      0.93       540
weighted avg       0.94      0.94      0.94       540
```

[14]:

```python
sklearn_digit = GradientBoostingClassifier(n_estimators=200,learning_rate = 0.
↪1,max_depth=1)

sklearn_digit.fit(X_train, y_train)
yhat_sk_digit = sklearn_digit.predict(X_test)
print("Sklearn classification report:\n ", classification_report(y_test,␣
↪yhat_sk_digit))
```

```
Sklearn classification report:
               precision    recall  f1-score   support

           0       1.00      0.98      0.99        56
           1       0.93      0.96      0.94        69
           2       1.00      1.00      1.00        52
           3       0.98      0.98      0.98        47
           4       0.97      0.97      0.97        63
           5       0.90      0.90      0.90        40
           6       0.98      0.94      0.96        64
           7       0.98      0.98      0.98        52
           8       0.93      0.93      0.93        55
           9       0.91      0.95      0.93        42

    accuracy                           0.96       540
   macro avg       0.96      0.96      0.96       540
weighted avg       0.96      0.96      0.96       540
```

[ ]:

```python

```