<https://www.datatechnotes.com/2019/06/regression-example-with-xgbregressor-in.html>

Regression Example with XGBRegressor in Python

   XGBoost stands for "Extreme Gradient Boosting" and it is an implementation of gradient boosting trees algorithm. The XGBoost is a popular supervised machine learning model with characteristics like computation speed, parallelization, and performance. You can find more about the model in this [link](https://xgboost.readthedocs.io/en/latest/index.html).  
  
   In this post, we'll learn how to define the XGBRegressor model and predict regression data in Python. The tutorial covers:

1. Preparing the data
2. Defining and fitting the model
3. Predicting and checking the results
4. Video tutorial
5. Source code listing

   We'll start by loading the required libraries. You may need to install them if they are not available on your machine.

**import** **xgboost** **as** **xgb**

**from** **sklearn.datasets** **import** load\_boston

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.model\_selection** **import** cross\_val\_score, KFold

**from** **sklearn.metrics** **import** mean\_squared\_error

**import** **matplotlib.pyplot** **as** **plt**

**Preparing data**  
  
   We use Boston house-price dataset as a regression dataset in this tutorial. After loading the dataset, first, we'll separate data into x - feature and y - label. Then we'll split them into the train and test parts. Here, I'll extract 15 percent of the dataset as test data.

boston = load\_boston()

x, y = boston.data, boston.target

xtrain, xtest, ytrain, ytest=train\_test\_split(x, y, test\_size=0.15)

**Defining and fitting the model**  
  
   For the regression problem, we'll use the XGBRegressor class of the xgboost package and we can define it with its default parameters. You can also set the new parameter values according to your data characteristics.

xgbr = xgb.XGBRegressor(verbosity=0)

print(xgbr)

XGBRegressor(base\_score=0.5, booster='gbtree', colsample\_bylevel=1,

colsample\_bynode=1, colsample\_bytree=1, gamma=0,

importance\_type='gain', learning\_rate=0.1, max\_delta\_step=0,

max\_depth=3, min\_child\_weight=1, missing=None, n\_estimators=100,

n\_jobs=1, nthread=None, objective='reg:linear', random\_state=0,

reg\_alpha=0, reg\_lambda=1, scale\_pos\_weight=1, seed=None,

silent=None, subsample=1, verbosity=1)

Next, we'll fit the model with train data.

xgbr.fit(xtrain, ytrain)

**Predicting and checking the results**

After training the model, we'll check the model training score.

score = xgbr.score(xtrain, ytrain)

print("Training score: ", score)

Training score: 0.9738225090795732

We can also apply the cross-validation method to evaluate the training score.

scores = cross\_val\_score(xgbr, xtrain, ytrain,cv=10)

print("Mean cross-validation score: %.2f" % scores.mean())

Mean cross-validataion score: 0.87

Or if you want to use the KFlold method in cross-validation it goes as below.

kfold = KFold(n\_splits=10, shuffle=**True**)

kf\_cv\_scores = cross\_val\_score(xgbr, xtrain, ytrain, cv=kfold )

print("K-fold CV average score: %.2f" % kf\_cv\_scores.mean())

K-fold CV average score: 0.87

Both methods show that the model is around 87 %  accurate on average.  
  
Next, we can predict test data, then check the prediction accuracy. Here, we'll use MSE and RMSE as accuracy metrics.

ypred = xgbr.predict(xtest)

mse = mean\_squared\_error(ytest, ypred)

print("MSE: %.2f" % mse)

MSE: 3.35

print("RMSE: %.2f" % (mse\*\*(1/2.0)))

RMSE: 1.83

Finally, we'll visualize the original and predicted test data in a plot to compare visually.

x\_ax = range(len(ytest))

plt.plot(x\_ax, ytest, label="original")

plt.plot(x\_ax, ypred, label="predicted")

plt.title("Boston test and predicted data")

plt.legend()

plt.show()

[Chart, histogram

Description automatically generated](https://1.bp.blogspot.com/-T4RAR8dKn5g/Xuw85BxWr7I/AAAAAAAAA34/jyz0_UAn_A4nqzXvauTdZ6B4xlW1W118wCLcBGAsYHQ/s1600/xgbr_prediction.png)

   In this post, we've briefly learned how to build the XGBRegressor model and predict regression data in Python. The full source code is listed below.

VIDEO:

<https://www.youtube.com/watch?v=-D2Px4b0XQE&t=1s>

**Source code listing** 

**import** **xgboost** **as** **xgb**

**from** **sklearn.datasets** **import** load\_boston

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.model\_selection** **import** cross\_val\_score, KFold

**from** **sklearn.metrics** **import** mean\_squared\_error

**import** **matplotlib.pyplot** **as** **plt**

boston = load\_boston()

x, y = boston.data, boston.target

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.15)

xgbr = xgb.XGBRegressor(verbosity=0)

print(xgbr)

xgbr.fit(xtrain, ytrain)

score = xgbr.score(xtrain, ytrain)

print("Training score: ", score)

*# - cross validataion*

scores = cross\_val\_score(xgbr, xtrain, ytrain, cv=5)

print("Mean cross-validation score: %.2f" % scores.mean())

kfold = KFold(n\_splits=10, shuffle=**True**)

kf\_cv\_scores = cross\_val\_score(xgbr, xtrain, ytrain, cv=kfold )

print("K-fold CV average score: %.2f" % kf\_cv\_scores.mean())

ypred = xgbr.predict(xtest)

mse = mean\_squared\_error(ytest, ypred)

print("MSE: %.2f" % mse)

print("RMSE: %.2f" % (mse\*\*(1/2.0)))

x\_ax = range(len(ytest))

plt.scatter(x\_ax, ytest, s=5, color="blue", label="original")

plt.plot(x\_ax, ypred, lw=0.8, color="red", label="predicted")

plt.legend()

plt.show()