Gboost_arodriguezsans_Mad

May 18, 2021

1 Madrid

1.1 Gradient Boosting Trees

1.1.1 Gradient Boosting Regressor

[1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ParameterGrid
from sklearn.preprocessing import MinMaxScaler
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_absolute_error, mean_squared_error

import multiprocessing

import warnings
warnings.filterwarnings('once')
```

[2]:
```
df_total = pd.read_excel('Total.xls')
# Edit columns names + Lower case column names
df_total.columns = map(str.lower, df_total.columns)
df_total.columns
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[2]:
```
Index(['sub_region_2', 'fecha', 'provincia_iso', 'num_casos.x',
       'num_casos_prueba_pcr', 'num_casos_prueba_test_ac',
       'num_casos_prueba_ag', 'num_casos_prueba_elisa',
       'num_casos_prueba_desconocida', 'num_casos.y', 'num_hosp', 'num_uci',
       'num_def', 'retail_and_recreation_percent_change_from_baseline',
       'grocery_and_pharmacy_percent_change_from_baseline',
       'parks_percent_change_from_baseline',
       'transit_stations_percent_change_from_baseline',
       'workplaces_percent_change_from_baseline',
       'residential_percent_change_from_baseline', 'total'],
      dtype='object')
```

[3]:
```
Mad = df_total.loc[df_total['sub_region_2'] == 'Madrid']
```

[4]:
```
# Set index
Mad = Mad.set_index('fecha')
```

[5]:
```
# We select columns of interest (mobility ones)
Mad = Mad[['num_casos.x']+['num_casos_prueba_pcr']+ list(Mad.loc[:
↪,'retail_and_recreation_percent_change_from_baseline':'total'])]
```

[6]:
```
# We create train and test datasets as in previous scenarios
X_train, X_test, y_train, y_test = train_test_split( #Mad,
                                                    Mad.drop(columns =␣
↪'num_casos.x'),
                                                    Mad['num_casos.x'],
                                                    shuffle = False, stratify =␣
↪None,
                                                    train_size=0.942)
```

[7]:
```
type(y_test)
```

[7]: pandas.core.series.Series

[8]:
```
type(X_test)
```

[8]: pandas.core.frame.DataFrame

[9]:
```
#X_test
```

[10]:
```
# Model generation
model = GradientBoostingRegressor(n_estimators = 10,
                                  loss         = 'ls',
                                  max_features = 'auto',
                                  random_state = 123)

model.fit(X_train, y_train)
```

```
# Prediction
predictions = model.predict(X = X_test)
rmse = mean_squared_error(y_test, predictions, squared = False)
print(f" RMSE: {rmse}")
```

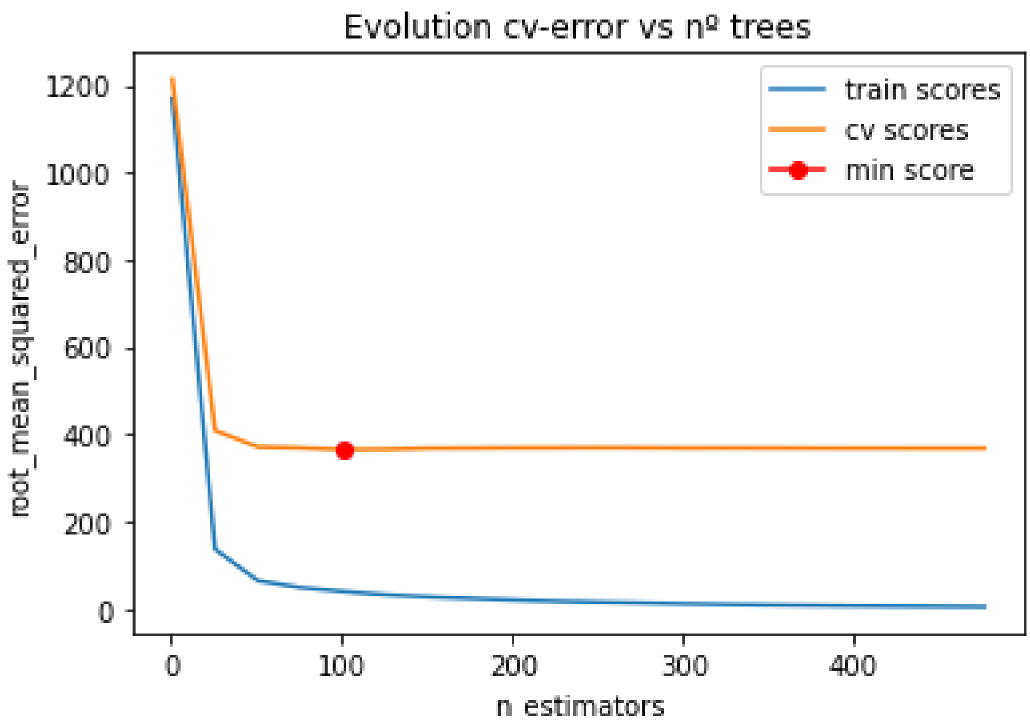
```
RMSE: 562.5284936344999
```

[11]:
```
# Validation with k-cross-validation and neg_root_mean_squared_error
train_scores = []
cv_scores    = []

# Values used
estimator_range = range(1, 500, 25)

# Train esach model with each values for n_estimators and extract its error
# test and k-cross-validation.
for n_estimators in estimator_range:
    model = GradientBoostingRegressor(
                n_estimators = n_estimators,
                loss         = 'ls',
                max_features = 'auto',
                random_state = 123)

    # Error train
    model.fit(X_train, y_train)
    predictions = model.predict(X = X_train)
    rmse = mean_squared_error(
            y_true  = y_train,
            y_pred  = predictions,
            squared = False
           )
    train_scores.append(rmse)
    # Error cv
    scores = cross_val_score(
                estimator = model,
                X         = X_train,
                y         = y_train,
                scoring   = 'neg_root_mean_squared_error',
                cv        = 5,
                n_jobs    = multiprocessing.cpu_count() - 1,
             )
    # aggregate scores cross_val_score() and pass to possitive
    cv_scores.append(-1*scores.mean())

# plot error evolution
fig, ax = plt.subplots(figsize=(6, 4))
```

```
ax.plot(estimator_range, train_scores, label="train scores")
ax.plot(estimator_range, cv_scores, label="cv scores")
ax.plot(estimator_range[np.argmin(cv_scores)], min(cv_scores),
        marker='o', color = "red", label="min score")
ax.set_ylabel("root_mean_squared_error")
ax.set_xlabel("n_estimators")
ax.set_title("Evolution cv-error vs nº trees")
plt.legend();
print(f"Optimal n_estimators: {estimator_range[np.argmin(cv_scores)]}")
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

```
Optimal n_estimators: 101
```



[12]:
```
# Validation k-cross-validation and neg_root_mean_squared_error
results = {}

# Values used
learning_rates = [0.001, 0.01, 0.1]
```

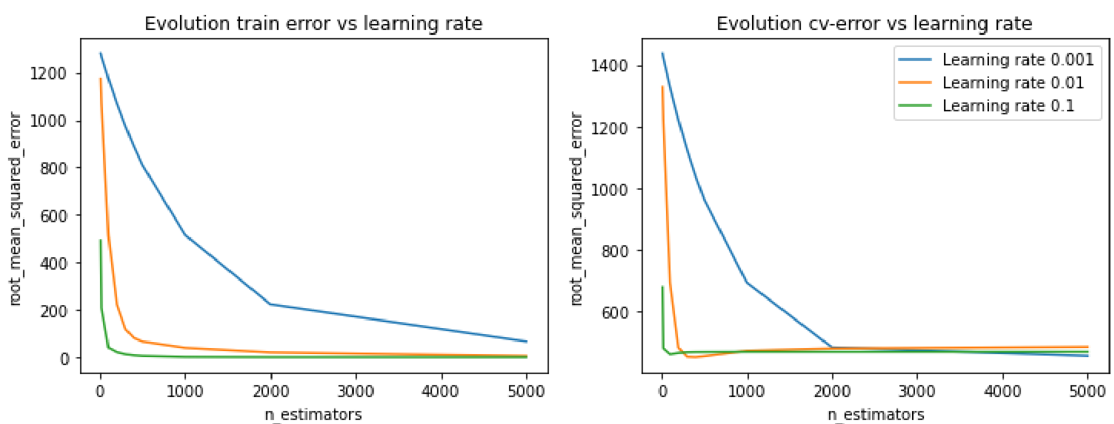
```
n_estimators   = [10, 20, 100, 200, 300, 400, 500, 1000, 2000, 5000]

# model train for each combination of learning_rate + n_estimator
# we get the error for tain and k-cross-validation.
for learning_rate in learning_rates:
    train_scores = []
    cv_scores    = []

    for n_estimator in n_estimators:

        model = GradientBoostingRegressor(
                    n_estimators  = n_estimator,
                    learning_rate = learning_rate,
                    loss          = 'ls',
                    max_features  = 'auto',
                    random_state  = 123
                 )

        # Error train
        model.fit(X_train, y_train)
        predictions = model.predict(X = X_train)
        rmse = mean_squared_error(
                y_true  = y_train,
                y_pred  = predictions,
                squared = False
               )
        train_scores.append(rmse)

        # Error CV
        scores = cross_val_score(
                    estimator = model,
                    X         = X_train,
                    y         = y_train,
                    scoring   = 'neg_root_mean_squared_error',
                    cv        = 3,
                    n_jobs    = multiprocessing.cpu_count() - 1
                 )
        # aggregate scores cross_val_score() and pass to possitive
        cv_scores.append(-1*scores.mean())

    results[learning_rate] = {'train_scores': train_scores, 'cv_scores':␣
 ↪cv_scores}
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
```

```
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[13]:
```python
# plot error evolution
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(12, 4))

for key, value in results.items():
    axs[0].plot(n_estimators, value['train_scores'], label=f"Learning rate␣
 ↪{key}")
    axs[0].set_ylabel("root_mean_squared_error")
    axs[0].set_xlabel("n_estimators")
    axs[0].set_title("Evolution train error vs learning rate")

    axs[1].plot(n_estimators, value['cv_scores'], label=f"Learning rate {key}")
    axs[1].set_ylabel("root_mean_squared_error")
    axs[1].set_xlabel("n_estimators")
    axs[1].set_title("Evolution cv-error vs learning rate")
    plt.legend();
```

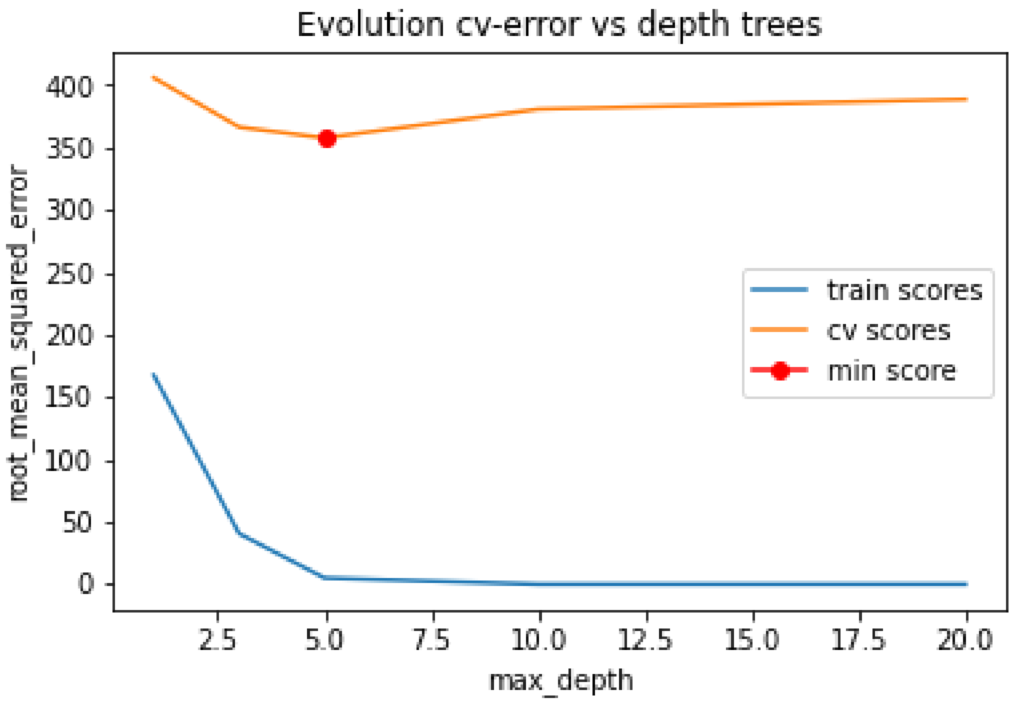
```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



[14]:
```python
# Validation k-cross-validation and neg_root_mean_squared_error
train_scores = []
cv_scores    = []

# Values used
max_depths = [1, 3, 5, 10, 20]
```

```python
# Train model for each max_depth
for max_depth in max_depths:

    model = GradientBoostingRegressor(
                n_estimators = 100,
                loss         = 'ls',
                max_depth    = max_depth,
                max_features = 'auto',
                random_state = 123
             )

    # Error train
    model.fit(X_train, y_train)
    predictions = model.predict(X = X_train)
    rmse = mean_squared_error(
            y_true  = y_train,
            y_pred  = predictions,
            squared = False
           )
    train_scores.append(rmse)

    # Error CV
    scores = cross_val_score(
                estimator = model,
                X         = X_train,
                y         = y_train,
                scoring   = 'neg_root_mean_squared_error',
                cv        = 5,
                n_jobs    = multiprocessing.cpu_count() - 1
             )
    # aggregate scores cross_val_score() pass to possitve
    cv_scores.append(-1*scores.mean())

# plots erros evolution
fig, ax = plt.subplots(figsize=(6, 3.84))
ax.plot(max_depths, train_scores, label="train scores")
ax.plot(max_depths, cv_scores, label="cv scores")
ax.plot(max_depths[np.argmin(cv_scores)], min(cv_scores),
        marker='o', color = "red", label="min score")
ax.set_ylabel("root_mean_squared_error")
ax.set_xlabel("max_depth")
ax.set_title("Evolution cv-error vs depth trees")
plt.legend();
print(f"Optimal max_depth: {max_depths[np.argmin(cv_scores)]}")
```

```
Optimal max_depth: 5
```



[15]:

```python
# Grid hyperparmeters
param_grid = {'max_features'  : ['auto', 'sqrt', 'log2'],
              'max_depth'     : [None, 1, 3, 5, 10, 20],
              'subsample'     : [0.5, 1],
              'learning_rate' : [0.001, 0.01, 0.1]
             }

# Grid-search with cv
grid = GridSearchCV(
        estimator  = GradientBoostingRegressor(
                        n_estimators        = 1000,
                        random_state        = 123,
                        # Early stop #
                        validation_fraction = 0.1,
                        n_iter_no_change    = 5,
                        tol                 = 0.0001
                    ),
        param_grid = param_grid,
        scoring    = 'neg_root_mean_squared_error',
        n_jobs     = multiprocessing.cpu_count() - 1,
        cv         = RepeatedKFold(n_splits=3, n_repeats=1, random_state=123),
        refit      = True,
        verbose    = 0,
        return_train_score = True
```

```
    )

grid.fit(X = X_train, y = y_train)

# Results
results = pd.DataFrame(grid.cv_results_)
results.filter(regex = '(param.*|mean_t|std_t)') \
    .drop(columns = 'params') \
    .sort_values('mean_test_score', ascending = False) \
    .head(4)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipykernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[15]:

```
   param_learning_rate param_max_depth param_max_features param_subsample  \
96                 0.1              10               auto             0.5
66                0.01              20               auto             0.5
36                0.01            None               auto             0.5
60                0.01              10               auto             0.5

    mean_test_score  std_test_score  mean_train_score  std_train_score
96      -153.892801       21.770760        -46.861491        14.652857
66      -159.492211       22.708394        -40.654660         8.429635
36      -160.517366       23.608976        -41.395662         7.584250
60      -160.936200       19.172829        -42.551761         9.385053
```

[16]:

```
# Best hyperparameters by cv
print("--------------------------")
print("Best hyperparameters by cv")
print("--------------------------")
print(grid.best_params_, ":", grid.best_score_, grid.scoring)
```

```
--------------------------
Best hyperparameters by cv
--------------------------
{'learning_rate': 0.1, 'max_depth': 10, 'max_features': 'auto', 'subsample':
0.5} : -153.89280064943958 neg_root_mean_squared_error
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipykernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[17]:
```
# Error test
model = grid.best_estimator_
predictions = model.predict(X = X_test)
rmse = mean_squared_error(
        y_true  = y_test,
        y_pred  = predictions,
        squared = False
       )
print(f"rmse test: {rmse}")
```

```
rmse test: 360.10492405051895
```

[18]:
```
importance_predictors = pd.DataFrame({#'predictor': Mad.columns,
                                    'predictor': Mad.drop(columns = 'num_casos.
 ↪x').columns,
                                      'importance': model.feature_importances_})
print("Importance of predictors")
print("------------------------")
importance_predictors.sort_values('importance', ascending=False)
```

```
Importance of predictors
------------------------

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[18]:

	predictor	importance
0	num_casos_prueba_pcr	0.960234
2	grocery_and_pharmacy_percent_change_from_baseline	0.011560
4	transit_stations_percent_change_from_baseline	0.008093
7	total	0.005679
3	parks_percent_change_from_baseline	0.004844
5	workplaces_percent_change_from_baseline	0.003793
6	residential_percent_change_from_baseline	0.003729
1	retail_and_recreation_percent_change_from_base…	0.002068

[19]:
```
importance = permutation_importance(
                estimator    = model,
                X            = X_train,
                y            = y_train,
                n_repeats    = 5,
                scoring      = 'neg_root_mean_squared_error',
                n_jobs       = multiprocessing.cpu_count() - 1,
                random_state = 123
```

```
            )

# Store results (mean / sd)
df_importance = pd.DataFrame(
                    {k: importance[k] for k in ['importances_mean',␣
 ↪'importances_std']}
                )
df_importance['feature'] = X_train.columns
df_importance.sort_values('importances_mean', ascending=False)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[19]:

```
   importances_mean  importances_std  \
0       1718.513026        29.030719
2        120.063644         9.928246
4        110.613417        15.190734
5         24.143391         2.841840
7         21.168309         1.405737
6         20.909076         2.817989
3         17.615066         1.562914
1         16.248805         2.972726

                                              feature
0                                 num_casos_prueba_pcr
2  grocery_and_pharmacy_percent_change_from_baseline
4      transit_stations_percent_change_from_baseline
5            workplaces_percent_change_from_baseline
7                                                total
6           residential_percent_change_from_baseline
3                 parks_percent_change_from_baseline
1  retail_and_recreation_percent_change_from_base…
```

[20]:

```
# Calculate the mean absolute error (MAE)
mae = mean_absolute_error(predictions, y_test)
print('MAE: ' + str(round(mae, 5)))

# Calculate the root mean squarred error (RMSE)
rmse = np.sqrt(mean_squared_error(y_test,predictions))
print('RMSE: ' + str(round(rmse, 5)))
```

```
MAE: 243.65012
RMSE: 360.10492
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[21]:
```
predictions_df = pd.DataFrame(predictions)
predictions_df.rename(columns={0:'Pred'},inplace=True)
y_test_df=pd.DataFrame(y_test)
y_test_df.reset_index(drop=True, inplace=True)
y_test_df
predictions_df['yt']=y_test_df['num_casos.x']
predictions_df
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

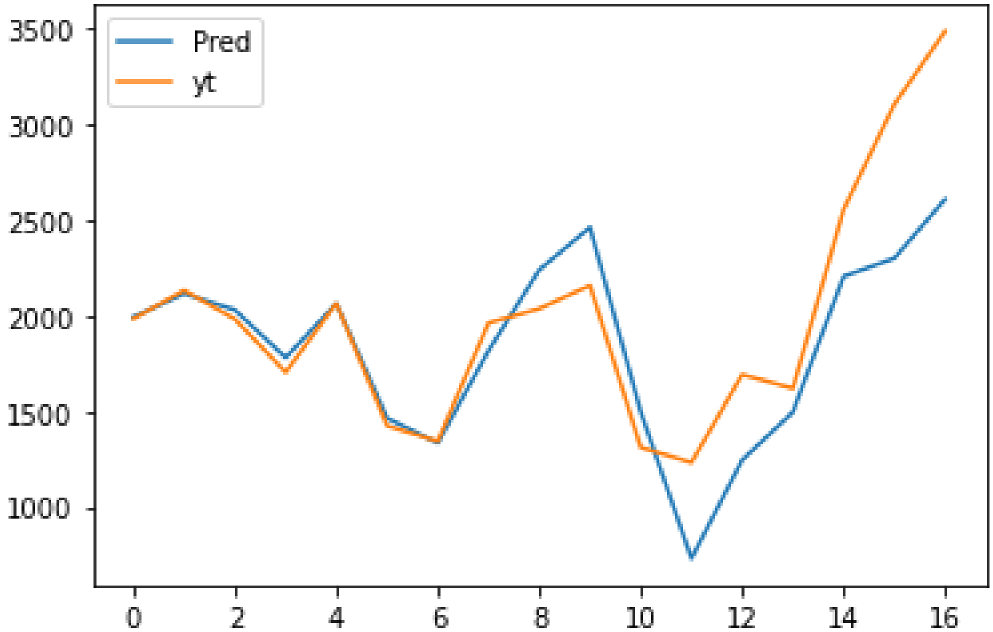
[21]:

	Pred	yt
0	1992.991694	1984
1	2114.252481	2132
2	2029.965570	1982
3	1780.724609	1704
4	2062.657047	2064
5	1465.788308	1426
6	1335.668969	1345
7	1816.776859	1962
8	2239.155601	2036
9	2461.312924	2158
10	1499.135737	1313
11	731.653480	1234
12	1246.896477	1692
13	1497.182561	1620
14	2204.925783	2555
15	2299.847753	3105
16	2608.160964	3485

[22]:
```
_ = predictions_df.plot()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
```

```
and should_run_async(code)
```



1.2 XGboost (Supervised)

Brownlee, J., 2021. How to Use XGBoost for Time Series Forecasting. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/xgboost-for-time-series-forecasting/ [Accessed 17 May 2021].

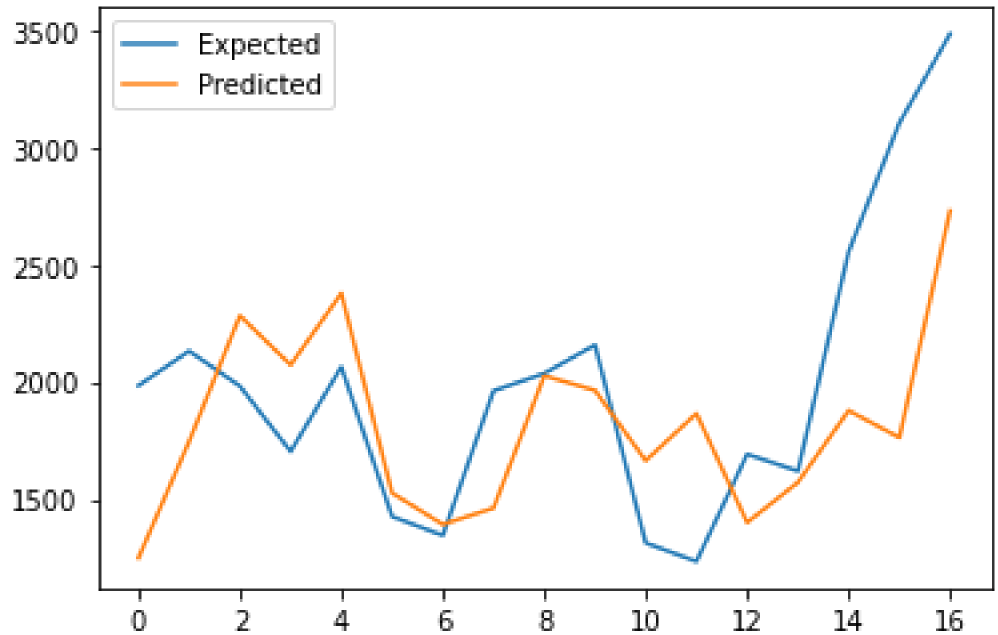
[23]:
```
from numpy import asarray
from pandas import DataFrame
from pandas import concat
from xgboost import XGBRegressor
from matplotlib import pyplot

# transform a time series dataset into a supervised learning dataset
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
        n_vars = 1 if type(data) is list else data.shape[0]
        df = DataFrame(data)
        cols = list()
        # input sequence (t-n, ... t-1)
        for i in range(n_in, 0, -1):
                cols.append(df.shift(i))
        # forecast sequence (t, t+1, ... t+n)
        for i in range(0, n_out):
                cols.append(df.shift(-i))
        # put it all together
```

```
        agg = concat(cols, axis=1)
        # drop rows with NaN values
        if dropnan:
                agg.dropna(inplace=True)
        return agg.values

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
        return data[:-n_test, :], data[-n_test:, :]

# fit an xgboost model and make a one step prediction
def xgboost_forecast(train, testX):
        # transform list into array
        train = asarray(train)
        # split into input and output columns
        trainX, trainy = train[:, :-1], train[:, -1]
        # fit model
        model = XGBRegressor(objective='reg:squarederror', n_estimators=1000)
        model.fit(trainX, trainy)
        # make a one-step prediction
        yhat = model.predict(asarray([testX]))
        return yhat[0]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test):
        predictions = list()
        # split dataset
        train, test = train_test_split(data, n_test)
        # seed history with training dataset
        history = [x for x in train]
        # step over each time-step in the test set
        for i in range(len(test)):
                # split test row into input and output columns
                testX, testy = test[i, :-1], test[i, -1]
                # fit model on history and make a prediction
                yhat = xgboost_forecast(history, testX)
                # store forecast in list of predictions
                predictions.append(yhat)
                # add actual observation to history for the next loop
                history.append(test[i])
                # summarize progress
                print('>expected=%.1f, predicted=%.1f' % (testy, yhat))
        # estimate prediction error
        error = mean_squared_error(test[:, -1], predictions, squared = False)
    #error = mean_absolute_error(test[:, -1], predictions)
        return error, test[:, -1], predictions
```

```python
# load the dataset
values = Mad['num_casos.x'].values
# transform the time series data into supervised learning
data = series_to_supervised(values, n_in=14)
# evaluate
mae, y, yhat = walk_forward_validation(data,17)
#print('MAE: %.3f' % mae)
print('RMSE: %.3f' % mae)
# plot expected vs preducted
plt.plot(y, label='Expected')
plt.plot(yhat, label='Predicted')
plt.legend()
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
C:\ProgramData\Anaconda3\lib\site-packages\xgboost\data.py:119: UserWarning: Use
subset (sliced data) of np.ndarray is not recommended because it will generate
extra copies and increase memory consumption
  warnings.warn(
```

```
>expected=1984.0, predicted=1248.7
>expected=2132.0, predicted=1747.0
>expected=1982.0, predicted=2282.9
>expected=1704.0, predicted=2072.6
>expected=2064.0, predicted=2378.3
>expected=1426.0, predicted=1526.3
>expected=1345.0, predicted=1392.2
>expected=1962.0, predicted=1462.0
>expected=2036.0, predicted=2026.3
>expected=2158.0, predicted=1963.9
>expected=1313.0, predicted=1664.4
>expected=1234.0, predicted=1866.0
>expected=1692.0, predicted=1400.4
>expected=1620.0, predicted=1571.6
>expected=2555.0, predicted=1878.4
>expected=3105.0, predicted=1763.0
>expected=3485.0, predicted=2730.2
RMSE: 528.761
```


3500
3000
2500
2000
1500
0
2
4
6
8
10
12
14
16
Expected
Predicted