Titulo: Exercício 5 - Desafio de regressores
Autor: Juan Sebastián Beleño Díaz
Data: 8 de Novembro de 2016

Introdução

Neste trabalho é feita uma comparação entre dois regressores (Gradient Boosting Regression e Random Forest Regression)usando a métrica de Mean Absolute Error (MAE). Todos os regressores usam optimização de hiperpârametros e validação externa. Finalmente, o melhor regressor é usado sobre os dados de teste para achar o valor de regressão, que será avaliado com a métrica MAE pelo professor Jaques Wainer.

Dados

Os arquivos utilizados neste trabalho são os dados de treino e os dados de teste. A primeira coluna dos dados de treino são os dados a serem calculados pelo regressor. O arquivo do conjunto de teste não tem o valor de regressão e deve ser calculado.

Preparação dos dados

Antes de começar trablahar com os dados é preciso incluir as dependecias do projeto:

```
# Loading the libraries
import math
import numpy as np
import pandas as pd

from sklearn import preprocessing
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_absolute_error
```

Existem muitas maneiras de abrir os arquivos e obter os dados, mas neste caso foi usado *pandas* para obter os dataframes diretamente desde a URL.

```
output_file = '../results/values_test.csv'

# Defining the URIs with raw data
url_train_data = 'http://www.ic.unicamp.br/%7Ewainer/cursos/2s2016/ml/train.csv'
url_test_data = 'http://www.ic.unicamp.br/%7Ewainer/cursos/2s2016/ml/test.csv'

# Reading the files with the raw data
df_train = pd.read_csv(url_train_data, header = None, delimiter = ",")
df_test = pd.read_csv(url_test_data, header = None, delimiter = ",")
```

Pré-processamento dos dados

O pré-processamento é utilizado para melhorar a precisão dos regressores. Neste projeto foram tranformados os dados categóricos em dados numéricos e não foi executado um PCA sobre os dados porque foi considerado que possívelmente isto podería dificultar o trabalho dos regressores. No entanto, é possível eliminar os dados com pouca variância.

```
# Creating a label encoders to handle categorical data
categorical_attributes = [4,5,6,7,8,9,11,12,15,16,17,20,22,28,29,30]
general_le = []
invert_index_le = 0
general_le_test = []
invert_index_le_test = 0

train_params = df_train.iloc[:, 1:33]
train_values = np.ravel(df_train.iloc[:, 1:2])

df_train_with_numbers = df_train
df_test_with_numbers = df_test

# Training set
for i in categorical_attributes:
    general_le.append(preprocessing.LabelEncoder())
    df_train_with_numbers[i] =
general_le[invert_index_le].fit_transform(df_train_with_numbers[i])
    invert_index_le = invert_index_le + 1

train_params_with_numbers = df_train_with_numbers.iloc[:, 1:33]

# Test set
for i in categorical_attributes:
    general_le_test.append(preprocessing.LabelEncoder())
    df_test_with_numbers[i-1] =
general_le_test[invert_index_le_test].fit_transform(df_test_with_numbers[i-1])
    invert_index_le_test = invert_index_le_test + 1

test_params_with_numbers = df_test_with_numbers


# Number of columns and rows in the train data
n_columns = df_train.shape[1] # 33
n_rows = df_train.shape[0] # 9000
```

Parâmetros

De maneira geral é definido um conjunto de variáveis que serão utilizadas para achar os melhores hiperparâmetros dos regressores. Além disso, é definido o número de splits na *cross-validation*.

```
# Number of splits for internal and external cross validation
n_internal_folds = 3
n_external_folds = 3
```

```
# Random Forest
best_external_n_estimators = 64
best_external_mae_rf = 1.0

# Gradient Boosting Descent
best_external_n_trees = 100
best_learning_rate = 0.1
best_external_mae_gbm = 1.0

# WARNING: I work with an i5 with 4 cores 3.3.GHz, please adjust this parameter
# to the number of cores your processor have
n_jobs = 4
pre_dispatch = 6 # 2 * n_jobs
```

Regressores

Os regressores escolhidos foram o Gradient Boosting Regressor e Random Forest Regressor porque eles trabalham bem com dados categóricos. No entanto, eles precisam de um tempo maior de processamento e mais memória por ser *ensembles*. A optimização de hiperparâmetros foi feita manualmente porque o pacote *GridSearchCV* de *sklearn* utiliza muita memória para o caso de regressores. 12GB de RAM não foram suficiêntes para executar um gridSearch com 3 cross-validation num conjunto de dados de 9000 filas e 33 colunas.

```
def rf_model(train_params, test_params, train_values, test_values):

    # [1] Oshiro, Thais Mayumi, Pedro Santoro Perez, and José Augusto Baranauskas.
    # "How many trees in a random forest?." International Workshop on Machine
    # Learning and Data Mining in Pattern Recognition. Springer Berlin Heidelberg,
2012.
    # Between 64 and 128 trees
    n_estimators_array = range(64,129, 8)
    best_n_estimators = 64
    best_mae = 1.0

    for n_estimators in n_estimators_array:

        regressor = RandomForestRegressor(n_estimators = n_estimators,
                                          criterion = 'mae',
                                          n_jobs = n_jobs)
        regressor.fit(train_params, train_values)

        model_predictions = regressor.predict(test_params)
        mae = mean_absolute_error(test_values, model_predictions)

        if mae < best_mae:
            best_mae = mae
            best_n_estimators = n_estimators

    return [best_mae, best_n_estimators]


def gbm_model(train_params, test_params, train_values, test_values):
```

```
    learning_rate_array = [0.1, 0.05]
    n_trees_array = [100, 125, 150]

    best_learning_rate = 0.1
    best_n_trees = 100
    best_mae = 1.0

    for learning_rate in learning_rate_array:
        for n_trees in n_trees_array:

            regressor = GradientBoostingRegressor(n_estimators = n_trees,
                                                  criterion = 'mae',
                                                  learning_rate = learning_rate)
            regressor.fit(train_params, train_values)

            model_predictions = regressor.predict(test_params)
            mae = mean_absolute_error(test_values, model_predictions)

            if mae < best_mae:
                best_mae = mae
                best_n_trees = n_trees
                best_learning_rate = learning_rate

    return [best_mae, best_n_trees,best_learning_rate]
```

Otimização de hiperparâmetros

A otimização dos hiperparâmetros usa uma cross validação externa de 3 folds para os regressores, obtendo também a métrica MAE para cada um deles. Ao final do loop for são obtidos os hiperparâmetros dos regressores.

```
# ------------------------ Here Goes the Magic -------------------------------

# Define the external K-Fold Stratified
external_skf = StratifiedKFold(n_splits = n_external_folds)
external_skf.get_n_splits(train_params, train_values)

# Iterate over external data
for external_train_index, external_test_index in external_skf.split(train_params,
train_values):

    # Split the external training set and the external test set
    external_params_train = train_params.iloc[external_train_index, :]
    external_params_train_with_numbers =
train_params_with_numbers.iloc[external_train_index, :]
    external_classes_train = train_values[external_train_index]
    external_params_test = train_params.iloc[external_test_index, :]
    external_params_test_with_numbers =
train_params_with_numbers.iloc[external_test_index, :]
    external_classes_test = train_values[external_test_index]

    # Random Forest Regressor
    rf_array = rf_model(external_params_train_with_numbers,
                        external_params_test_with_numbers,
```

```
                        external_classes_train,
                        external_classes_test)

    # Using external cross-validation to find the best hyperparameter for RF
    if rf_array[0] < best_external_mae_rf:
        best_external_mae_rf = rf_array[0]
        best_external_n_estimators = rf_array[1]

    # Gradient  Boosting Descent
    gbm_array = gbm_model(external_params_train_with_numbers,
                          external_params_test_with_numbers,
                          external_classes_train,
                          external_classes_test)

    # Using external cross-validation to fin best hyperparameters for GBM
    if gbm_array[0] < best_external_mae_gbm:
        best_external_mae_gbm = gbm_array[0]
        best_external_n_trees = gbm_array[1]
        best_learning_rate = gbm_array[2]
```

Resultados

O Random Forest Regressor foi o melhor regressor para este conjunto de dados com um MAE = 0.0 ou precisão perfeita. Os hiperparâmetros otimizados para o RF Regressor são n_estimators = 64. Provávelmente este regressor tem *overfitting* sobre os dados de treinamento. No entanto, ele é usado para calcular o resultado para os dados de teste e o professor será quem avalie a precisão do meu regressor.

```
print('Random Forest')
print('# of estimators: ', best_external_n_estimators)
print('MAE: ', best_external_mae_rf)
print("---------------------------------")
print('Gradient Boosting Descent')
print('# of estimators: ', best_external_n_trees)
print('Learning rate: ', best_learning_rate)
print('MAE: ', best_external_mae_gbm)
print("---------------------------------")
```

```
Random Forest
# of estimators:  64
MAE:  0.0
---------------------------------
Gradient Boosting Descent
# of estimators:  150
Learning rate:  0.1
MAE:  7.37929825042e-05
---------------------------------
```

Regressão dos dados de teste

O Random Forest Regressor com os hiperparâmetros otimizados cálcula os valores para o conjunto de teste.

```
regressor = RandomForestRegressor(n_estimators = best_external_n_estimators,
                                  criterion = 'mae',
                                  n_jobs = n_jobs)
regressor.fit(train_params_with_numbers, train_values)
value_predictions = regressor.predict(test_params_with_numbers)

df_predictions = pd.DataFrame(value_predictions)
df_predictions.to_csv(output_file, sep=',', encoding='utf-8', header = False,
index=False)
```