Université Paris Saclay

Lockdown Period Load Forecasting

Auteur :
Béreux Nicolas
Denis Matthieu

Superviseur :
Yannig Goude

13 mars 2021

Table des matières

1 Présentation du sujet

```
   tidyverse    lubridate       ranger       pracma      Metrics         mgcv
        TRUE         TRUE         TRUE         TRUE         TRUE         TRUE
       keras      visreg        caret         mc2d        opera        abind
        TRUE         TRUE         TRUE         TRUE         TRUE         TRUE
randomForest   tensorflow       plot3D        e1071          xts
        TRUE         TRUE         TRUE         TRUE         TRUE

$cross_validation.r
$cross_validation.r$value
function (data)
{
    set.seed(123)
    to.train = rbern(n = length(data$Load), p = 0.8) == T
    cross.train = data[to.train, ]
    cross.test = data[!to.train, ]
    return(list(train = cross.train, test = cross.test))
}

$cross_validation.r$visible
[1] FALSE


$days_to_numeric.r
$days_to_numeric.r$value
function (data)
{
    data$WeekDays[data$WeekDays == "Monday"] = 1
    data$WeekDays[data$WeekDays == "Tuesday"] = 2
    data$WeekDays[data$WeekDays == "Wednesday"] = 3
    data$WeekDays[data$WeekDays == "Thursday"] = 4
    data$WeekDays[data$WeekDays == "Friday"] = 5
    data$WeekDays[data$WeekDays == "Saturday"] = 6
    data$WeekDays[data$WeekDays == "Sunday"] = 7
    data$WeekDays = as.numeric(data$WeekDays)
    return(data$WeekDays)
}

$days_to_numeric.r$visible
[1] FALSE


$descriptive_analysis.r
$descriptive_analysis.r$value
function (ts)
{
    month = as.factor(.indexmon(ts))
    mean.month = tapply(ts, month, mean)
    noise = c()
    for (i in c(1:length(ts))) {
        noise = c(noise, mean.month[month[i]] - ts[i])
    }
    return(noise)
}

$descriptive_analysis.r$visible
[1] FALSE
```

```
$evaluate.r
$evaluate.r$value
function (test_label, predicted_set)
{
    return(rmse(test_label, predicted_set))
}

$evaluate.r$visible
[1] FALSE


$format_data.r
$format_data.r$value
function (file_train, file_test)
{
    print(paste("Load and format ", file_train, sep = " "))
    train_set = read_csv(file_train, col_types = cols())
    train_set$WeekDays[train_set$WeekDays == "Monday"] <- 1
    train_set$WeekDays[train_set$WeekDays == "Tuesday"] <- 2
    train_set$WeekDays[train_set$WeekDays == "Wednesday"] <- 3
    train_set$WeekDays[train_set$WeekDays == "Thursday"] <- 4
    train_set$WeekDays[train_set$WeekDays == "Friday"] <- 5
    train_set$WeekDays[train_set$WeekDays == "Saturday"] <- 6
    train_set$WeekDays[train_set$WeekDays == "Sunday"] <- 7
    train_set$WeekDays = as.integer(train_set$WeekDays)
    train_set$Year = NULL
    train_set$Date = NULL
    train_label = data.matrix(train_set$Load)
    train_set$Load = NULL
    train_set = data.matrix(train_set)
    print(paste("Load and format ", file_test, sep = " "))
    test_set = read_csv(file_test, col_types = cols())
    test_set$WeekDays[test_set$WeekDays == "Monday"] <- 1
    test_set$WeekDays[test_set$WeekDays == "Tuesday"] <- 2
    test_set$WeekDays[test_set$WeekDays == "Wednesday"] <- 3
    test_set$WeekDays[test_set$WeekDays == "Thursday"] <- 4
    test_set$WeekDays[test_set$WeekDays == "Friday"] <- 5
    test_set$WeekDays[test_set$WeekDays == "Saturday"] <- 6
    test_set$WeekDays[test_set$WeekDays == "Sunday"] <- 7
    test_set$WeekDays = as.integer(test_set$WeekDays)
    test_label = test_set$Load.1
    tmp = test_set$Load.1
    for (i in c(1:(length(tmp) - 1))) {
        test_label[i] = tmp[i + 1]
    }
    test_set$Year = NULL
    test_set$Date = NULL
    test_set$Id = NULL
    test_set$Usage = NULL
    test_set = data.matrix(test_set)
    test_set = data.matrix(test_set)
    return(list(train_set = train_set, train_label = train_label,
        test_set = test_set, test_label = test_label))
}

$format_data.r$visible
[1] FALSE
```

```
$fourier.r
$fourier.r$value
function (train, test, plt = FALSE)
{
    total.time = c(1:(nrow(train) + nrow(test)))
    length(total.time)
    train$time = total.time[1:nrow(train)]
    test$time = tail(total.time, nrow(test))
    fourier.make.matrix = function(t, k, period) {
        w = 2 * pi/period
        ret = cbind(cos(w * t), sin(w * t))
        for (i in c(2:K)) {
            ret = cbind(ret, cos(i * w * t), sin(i * w * t))
        }
        return(ret)
    }
    K = 5
    period = 365
    fourier.train = fourier.make.matrix(train$time, K, period)
    fourier.test = fourier.make.matrix(test$time, K, period)
    fourier.train.df = data.frame(train$Load, fourier.train)
    fourier.test.df = data.frame(fourier.test)
    reg = lm(train.Load ~ ., data = fourier.train.df)
    pred.fourier = predict(reg, newdata = fourier.test.df)
    total.fourier = c(reg$fitted, pred.fourier)
    if (plt) {
        par(mfrow = c(1, 1))
        plot(train$Load, type = "l", xlim = c(0, length(total.time)))
        lines(reg$fitted, col = "red", lwd = 2)
        lines(test$time, pred.fourier, col = "green", lwd = 2)
    }
    return(pred.fourier)
}

$fourier.r$visible
[1] FALSE


$GAM.r
$GAM.r$value
function (train, test, plt = FALSE)
{
    Gam <- gam(Load ~ s(Load.1) + s(Load.7) + s(Temp) + s(Temp_s95) +
        s(WeekDays, k = 7) + s(GovernmentResponseIndex), data = train)
    summary(Gam)
    gam.train = predict(Gam, newdata = train)
    gam.test = predict(Gam, newdata = test)
    if (plt) {
        par(mfrow = c(1, 1))
        plot(train$Load, type = "l", xlim = c(0, length(total.time)))
        lines(train$time, Gam$fit, col = "red", lwd = 1)
        lines(test$time, gam.test, col = "green", lwd = 1)
    }
    return(gam.test)
}

$GAM.r$visible
[1] FALSE
```

```
$lstm.r
$lstm.r$value
function (train, test, plt = FALSE)
{
    labels = train[, 2]
    train.lstm = train[, -c(1, 2, 22, 23)]
    test.lstm = test[, -c(1, 20, 21, 23, 24)]
    lstm = function(train_set, train_label, test_set) {
        window = 10
        n_windows = nrow(train_set) - window + 1
        n_features = ncol(train_set)
        x = array(data = NA, dim = c(n_windows, window, n_features))
        y = array(data = NA, dim = c(n_windows, window, 1))
        for (i in 1:n_windows) {
            x[i, , ] = data.matrix(train_set[i:(i + window -
                1), ])
            y[i, , ] = as.matrix(data.matrix(train_label[i:(i +
                window - 1), ]))
        }
        i = 1
        print(size(y))
        batch_size = 1
        lrelu = function(x) tf.keras.activations.relu(x, alpha = 0.1)
        model = keras_model_sequential() %>% layer_lstm(units = 64,
            batch_input_shape = c(batch_size, window, n_features),
            dropout = 0.2, recurrent_dropout = 0.2, return_sequences = T,
            ) %>% time_distributed(layer_dense(units = 1))
        model %>% compile(loss = "mae", optimizer = optimizer_rmsprop())
        model %>% fit(x, y, epochs = 15)
        pred.train = model %>% predict(X_train)
        pred.test = model %>% predict(X_test)
        return(list(train = pred.train, test = pred.test, model = model))
    }
    pred.lstm = lstm(train.lstm, labels, test.lstm)
    if (plt) {
        par(mfrow = c(1, 1))
        plot(train$Load, type = "l", xlim = c(0, length(total.time)))
        lines(test$time, pred.lstm$test, col = "green", lwd = 1)
    }
    N = length(test$Load.1)
    RMSE = rmse(pred.lstm$test[-N], test$Load.1[2:N])
    RMSE
    return(pred.lstm)
}

$lstm.r$visible
[1] FALSE


$main.r
NULL

$neural_network.r
$neural_network.r$value
function (train, test, plt = FALSE)
{
    labels = select(train, "Load")
    train.lstm = train
```

```
test.lstm = test
lstm = function(train_set, train_label, test_set) {
    window = nrow(test_set)
    n_windows = nrow(train_set) - window + 1
    n_features = ncol(train_set)
    x_train = array(data = NA, dim = c(n_windows, window,
        n_features))
    y_train = array(data = NA, dim = c(n_windows, window,
        1))
    for (i in window:nrow(train_set)) {
        x_train[i - window + 1, , ] = data.matrix(train_set[(i -
            window + 1):i, ])
        y_train[i - window + 1, , ] = as.matrix(data.matrix(train_label[(i -
            window + 1):i, ]))
    }
    x_test = array(data = NA, dim = c(1, window, n_features))
    for (i in window:nrow(test_set)) {
        x_test[i - window + 1, , ] = data.matrix(test_set[(i -
            window + 1):i, ])
    }
    batch_size = 40
    n = size(x_train)[1]
    to_remove = sample.int(n, n%%batch_size)
    x_train = x_train[-to_remove, , ]
    y_train = y_train[-to_remove, , ]
    y_train = array(y_train, dim = c(size(y_train)[1], size(y_train)[2],
        1))
    model = keras_model_sequential() %>% layer_lstm(units = 64,
        batch_input_shape = c(batch_size, window, n_features),
        return_sequences = T, activation = layer_activation_relu(max_value = 40000),
        kernel_regularizer = regularizer_l2(0.001), bias_regularizer = regularizer_l2(0.001)) %>%
        layer_lstm(units = 64, return_sequences = T, activation = layer_activation_relu(max_value
            kernel_regularizer = regularizer_l2(0.001), bias_regularizer = regularizer_l2(0.001))
        layer_lstm(units = n_features, return_sequences = T,
            activation = layer_activation_relu(max_value = 40000),
            kernel_regularizer = regularizer_l2(0.001), bias_regularizer = regularizer_l2(0.001))
        time_distributed(layer_dense(units = 1))
    model %>% compile(loss = "mse", optimizer_rmsprop(lr = 0.01,
        rho = 0.9, epsilon = NULL, decay = 0, clipnorm = 1,
        clipvalue = NULL))
    model %>% fit(x_train, y_train, epochs = 4, batch_size = batch_size)
    print("www")
    model_evaluate = keras_model_sequential() %>% layer_lstm(units = 64,
        batch_input_shape = c(1, window, n_features), return_sequences = T,
        activation = layer_activation_relu(max_value = 40000),
        kernel_regularizer = regularizer_l2(0.001), bias_regularizer = regularizer_l2(0.001)) %>%
        layer_lstm(units = 64, return_sequences = T, activation = layer_activation_relu(max_value
            kernel_regularizer = regularizer_l2(0.001), bias_regularizer = regularizer_l2(0.001))
        layer_lstm(units = n_features, return_sequences = T,
            activation = layer_activation_relu(max_value = 40000),
            kernel_regularizer = regularizer_l2(0.001), bias_regularizer = regularizer_l2(0.001))
        time_distributed(layer_dense(units = 1))
    print("www1")
    set_weights(model_evaluate, get_weights(model))
    print("www2")
    pred.test = model_evaluate %>% predict(x_test, batch_size = 1) %>%
        .[, , 1]
    print("www4")
    return(list(train = 1, test = pred.test, model = model_evaluate))
```

```
    }
    pred.lstm = lstm(train.lstm, labels, test.lstm)
    if (plt) {
        par(mfrow = c(1, 1))
        plot(train$Load, type = "l", xlim = c(0, length(total.time)))
        lines(test$time, pred.lstm$test, col = "green", lwd = 1)
    }
    N = length(test$Load.1)
    RMSE = rmse(pred.lstm$test[-N], test$Load.1[2:N])
    RMSE
    return(pred.lstm)
}

$neural_network.r$visible
[1] FALSE


$random_forest.r
$random_forest.r$value
function (train, test, n_trees, plt = FALSE)
{
    rf = randomForest(Load ~ ., data = train, mtry = n_trees,
        importance = TRUE, na.action = na.omit)
    pred.test.rf = predict(rf, test)
    print(rf$importance)
    if (plt) {
        par(mfrow = c(1, 1))
        plot(train$Load, type = "l", xlim = c(0, length(total.time)))
        lines(test$time, pred.test.rf, col = "green", lwd = 1)
    }
    N = length(test$Load.1)
    RMSE = rmse(pred.test.rf[-N], test$Load.1[2:N])
    RMSE
    return(list(pred.test.rf = pred.test.rf, rf = rf))
}

$random_forest.r$visible
[1] FALSE


$scale.r
$scale.r$value
function (scaled, scaler, feature_range = c(0, 1))
{
    min = scaler[1]
    max = scaler[2]
    t = length(scaled)
    mins = feature_range[1]
    maxs = feature_range[2]
    inverted_dfs = numeric(t)
    for (i in 1:t) {
        X = (scaled[i] - mins)/(maxs - mins)
        rawValues = X * (max - min) + min
        inverted_dfs[i] <- rawValues
    }
    return(inverted_dfs)
}

$scale.r$visible
```

```
[1] FALSE


$xgboost.r
$xgboost.r$value
function (train_set, train_label, test_set)
{
    param = list(booster = "gblinear", objective = "reg:squarederror",
        eval_metric = "rmse", lambda = 3e-04, alpha = 3e-04,
        nthread = 2, eta = 0.1)
    print("Model : XGBOOST")
    xgbmodel = xgboost(data = train_set, label = train_label,
        nrounds = 200, params = param, verbose = 0)
    pred = predict(xgbmodel, test_set)
    return(pred)
}

$xgboost.r$visible
[1] FALSE

[1] "Load and format  ./data/train_V2.csv"
[1] "Load and format  ./data/test_V2.csv"
```

Nous allons nous intéresser à la prédiction de la consommation électrique en France durant la pandémie de COVID-19. Cette situation de confinement et couvre-feu étant inédite, nous n'avons pas de données et de modèles adaptés. Nous allons donc essayer de fournir un modèle prédictif dont nous évaluerons la qualité en utilisant la Root Mean Squared Error (RMSE). Prédire la consommation électrique à l'avance permet d'adapter la production en amont et ainsi d'éviter des coupures de courant en cas de demande élevée et d'éviter d'en produire trop. Les centrales à énergie fossile (gaz, charbon) étant les plus rapides à mettre en marche et arrêter, on peut ainsi réduire au minimum l'émission de gaz à effet de serre.

Ce jeu de données comporte 3028 ligne pour la partie d'entrainement et 275 lignes pour la partie de test. Chaque ligne est composée de

```
     Index                          Load.1          Load.7           Temp
 Min.   :2012-01-01 00:00:00   Min.   :35589   Min.   :35589   Min.   :-4.897
 1st Qu.:2014-01-26 18:00:00   1st Qu.:46727   1st Qu.:46757   1st Qu.: 7.830
 Median :2016-02-22 12:00:00   Median :51261   Median :51319   Median :12.084
 Mean   :2016-02-22 12:00:00   Mean   :54617   Mean   :54669   Mean   :12.542
 3rd Qu.:2018-03-20 06:00:00   3rd Qu.:63140   3rd Qu.:63173   3rd Qu.:17.498
 Max.   :2020-04-15 01:00:00   Max.   :94097   Max.   :94097   Max.   :28.066
    Temp_s95          Temp_s99        Temp_s95_min      Temp_s95_max
 Min.   :-4.522   Min.   :-4.152   Min.   :-6.186   Min.   :-3.782
 1st Qu.: 7.824   1st Qu.: 7.890   1st Qu.: 6.704   1st Qu.: 8.933
 Median :12.076   Median :12.035   Median :10.717   Median :13.460
 Mean   :12.542   Mean   :12.541   Mean   :11.155   Mean   :13.935
 3rd Qu.:17.519   3rd Qu.:17.596   3rd Qu.:15.886   3rd Qu.:19.064
 Max.   :27.985   Max.   :26.318   Max.   :25.438   Max.   :30.514
  Temp_s99_min     Temp_s99_max          toy               WeekDays
 Min.   :-4.518   Min.   :-3.732   Min.   :0.001338   Min.   :1.000
 1st Qu.: 7.615   1st Qu.: 8.253   1st Qu.:0.230859   1st Qu.:2.000
 Median :11.652   Median :12.504   Median :0.481530   Median :4.000
 Mean   :12.175   Mean   :12.978   Mean   :0.487565   Mean   :3.999
 3rd Qu.:17.190   3rd Qu.:18.048   3rd Qu.:0.741110   3rd Qu.:6.000
 Max.   :25.630   Max.   :27.087   Max.   :0.998662   Max.   :7.000
       BH               Month              DLS         Summer_break
 Min.   :0.00000   Min.   : 1.000   Min.   :1.00   Min.   : 0.0000
 1st Qu.:0.00000   1st Qu.: 3.000   1st Qu.:1.00   1st Qu.: 0.0000
 Median :0.00000   Median : 6.000   Median :2.00   Median : 0.0000
 Mean   :0.03534   Mean   : 6.375   Mean   :1.57   Mean   : 0.9247
 3rd Qu.:0.00000   3rd Qu.: 9.000   3rd Qu.:2.00   3rd Qu.: 0.0000
```

```
Max.   :1.00000   Max.   :12.000   Max.   :2.00   Max.   :10.0000
Christmas_break   GovernmentResponseIndex
Min.   : 0.0000   Min.   : 0.000
1st Qu.: 0.0000   1st Qu.: 0.000
Median : 0.0000   Median : 0.000
Mean   : 0.9181   Mean   : 1.103
3rd Qu.: 0.0000   3rd Qu.: 0.000
Max.   :20.0000   Max.   :72.500
```

Les données d'entrainement vont du 1er janvier 2012 au 15 avril 2020 (soit 1 mois après le début du premier confinement). Nous vérifions ensuite si il y a des valeurs invalides dans notre jeu de données : ■apercu,echo=FALSE■= which(is.na(train$_set))which(is.na(test_set))Iln'yenaaucune.$

Nous allons maintenant nous concentrer uniquement sur les données d'entrainement.

2 Analyse descriptive des données

Commençons par regarder à quoi ressemble nos données

2.1 Saisonnalité

Nous observons une saisonnalité annuelle. Si nous nous concentrons sur deux mois : Nous observons ici une saisonnalité au niveau de la semaine. Vérifions cela en traçant les consommations moyennes sur tout le jeu de données Ces figures confirment une saisonnalité à l'année avec une consommation électrique plus élevée en hiver et chutant en été, particulièrement en août. Nous pouvons corréler cela avec la température : d'après EDF, le chauffage électrique correspond à 62% de la consommation électrique au sein d'une maison ou d'un appartement en France. Nous observons aussi une baisse de la consommation électrique lors du week-end (car moins de personnes travaillent le week-end).

2.2 Corrélation

2.3 Statistiques de base