Процедура отбора моделей MCS

Boris Demeshev

22 February 2016

Процедура MCS позволяет из множества моделей отбросить «плохие». По окончании процедуры отбора могут остаться неотброшенными несколько моделей.

Процедура выглядит так:

1. Для каждой модели рассчитывается её «степень несовершенства», v_i.
2. Проверяется гипотеза о том, что все модели одинаково «хороши».
3. Если гипотеза не отвергается, то процедура заканчивается. Если не все модели одинаково хороши, то самая плохая модель отбрасывается.
4. Переходим к шагу 1 с меньшим числом моделей.

Интуитивное пояснение:

На шаге 1 в качестве степени несовершенства используется один из двух показателей качества прогнозов. Можно рассчитать отставание модели по качеству прогнозов от самого сильного конкурента. Можно рассчитать усреднённое отставание модели от остальных.

На шаге 2 используют одну из трёх статистик: T_{max}, T_R, T_{SQ}. Статистики T_{max} и T_R измеряют, насколько плоха самая плохая из моделей. Статистика T_{SQ} показывает, насколько сильно разняться модели по качеству прогнозов. Распределение у всех трёх статистик не стандартное.

Для более подбробного изложения введём обозначения:

$l_{i,t}$ — штраф модели i за наблюдение t, например, квадратичный

$$l_{i,t} = (\hat{y}_t^{(i)} - y_t)^2,$$

где $\hat{y}_t^{(i)}$ — прогноз по i-ой модели.

$d_{ij,t} = l_{i,t} - l_{j,t}$ — отставание модели i от модели j в прогнозе наблюдения t. Если $d_{ij,t} > 0$, значит модель i прогнозирует момент времени t хуже модели j.

$\bar{d}_{ij} = \sum_t d_{ij,t}/T$ — отставание модели i от модели j по качеству прогнозов. Если $d_{ij} > 0$, значит модель i прогнозирует хуже модели j.

$\bar{d}_{i.} = \sum_j \bar{d}_{ij}/m$ — усреднённое отставание модели i от конкурентов.

$t_{i.} = \frac{\bar{d}_{i.}}{se(\bar{d}_{i.})}$, стандартная ошибка рассчитывается с помощью бутстрэпа. Можно интерпретировать как стандартизированное усреднённое отставание модели i от конкурентов.

$t_{ij} = \frac{\bar{d}_{ij}}{se(\bar{d}_{ij})}$, стандартная ошибка рассчитывается с помощью бутстрэпа. Можно интерпретировать как стандартизированное отставание модели i от модели модели j. Заметим, что $t_{ij} = -t_{ji}$ и $se(t_{ij}) = se(t_{ji})$.

Пакет MCS в R выводит результаты последней процедуры отбрасывания модели:

- v_M$_i = t_{i.}$. Усреднённая отставание модели i от конкурентов. Индекс M означает, что на основании v_M$_i$ считается статистика T_{max}.
- v_R$_i = \max_j t_{ij}$. Отставание модели i от самого сильного её конкурента. Индекс R означает, что на основании v_R$_i$ считается статистика T_R. Отрицательное значение возможно только у модели-лидера.

- $\text{Loss}_i = \sum_t l_{i,t}/T$ — среднее значение функции потерь модели i
- Rank_R_i — рейтинг модели i, рассчитанный с помощью v_R_i. Чем меньше, тем лучше.
- Rank_M_i — рейтинг модели i если считать с помощью v_M_i. Чем меньше, тем лучше.

$T_{max} = \max_i t_{i.} = \max_i \text{v\_i\_M}$ (R считает, matlab не считает)

$T_R = \max_{ij} t_{ij} = \max_i \text{v\_i\_R}$ (R и matlab)

$T_{SQ} = \sum_{i<j} t_{ij}^2$ (R не считает, matlab считает)

```
library("MCS")
data(Loss) # матрица $l_{i,t}$, в столбцах модели, в строках время
Loss_mini <- Loss[1:100, 20:30]
best_models <- MCSprocedure(Loss_mini, verbose = FALSE)
print(best_models)
```

```
##
## ------------------------------------------
## -          Superior Set of Models         -
## ------------------------------------------
##                 Rank_M         v_M MCS_M Rank_R        v_R MCS_R
## gjrGARCH-snorm      1 -1.90675693     1      1 -0.5700125     1
## gjrGARCH-sstd       3 -0.62128505     1      3  0.8515934     1
## gjrGARCH-sged       5  0.16491154     1      5  1.3680377     1
## gjrGARCH-jsu        4 -0.01786092     1      4  1.2451019     1
## gjrGARCH-ghyp       6  0.17612446     1      6  1.3762287     1
## apARCH-norm         2 -1.02346454     1      2  0.5696511     1
## apARCH-std          8  1.09438313     1      8  2.0094817     1
## apARCH-ged          9  1.41299427     1      9  2.2182017     0
## apARCH-snorm        7  0.74608482     1      7  1.7747139     1
##                        Loss
## gjrGARCH-snorm 0.0004316483
## gjrGARCH-sstd  0.0004326200
## gjrGARCH-sged  0.0004332168
## gjrGARCH-jsu   0.0004330774
## gjrGARCH-ghyp  0.0004332255
## apARCH-norm    0.0004322969
## apARCH-std     0.0004339199
## apARCH-ged     0.0004341585
## apARCH-snorm   0.0004336560
##
## Details
## ------------------------------------------
##
## Number of eliminated models  :   2
## Statistic    :   Tmax
## Elapsed Time :   Time difference of 27.09596 secs
```

Проблемы метода:

- Чувствительность к посторонним альтернативам

Например, алгоритм из трёх моделей может оставлять две, а добавишь в набор сравниваемых моделей ещё четыре модели хуже любой исходной, и ни одна не будет откинута!

Численный пример. Генерируем три примерно одинаковые модели:

```
n <- 100

set.seed(23)
true_y <- rep(0, n) + rnorm(n, sd = 0.0005)
m1 <- rep(0.1, n) + rnorm(n, sd = 0.0005)
m2 <- rep(0.10001, n) + rnorm(n, sd = 0.0005)
m3 <- rep(0.1001, n) + rnorm(n, sd = 0.0005)

# MCS with 3 models

forecasts <- cbind(m1, m2, m3)
n_models <- ncol(forecasts)
actual <- matrix(rep(true_y, n_models), ncol = n_models)

loss <- (actual - forecasts) ^ 2
best_models <- MCSprocedure(loss)
```

```
##
## Model m3 eliminated 2016-03-27 23:09:36
## ###########################################################################
## Superior Set Model created   :
##    Rank_M        v_M  MCS_M Rank_R        v_R  MCS_R        Loss
## m1      2  0.9979013 0.3022      2  0.9979013 0.3046 0.009990946
## m2      1 -0.9976900 1.0000      1 -0.9976900 1.0000 0.009987804
## p-value  :
## [1] 0.3022
##
## ###########################################################################
```

Добавляем ещё четыре заведомо более плохих модели:

```
m4 <- rep(0.25, n) + rnorm(n, sd = 0.0005)
m5 <- rep(0.25, n) + rnorm(n, sd = 0.0005)
m6 <- rep(0.25, n) + rnorm(n, sd = 0.0005)
m7 <- rep(0.25, n) + rnorm(n, sd = 0.0005)

# MCS with 7 models

forecasts <- cbind(m1, m2, m3, m4, m5, m6, m7)
n_models <- ncol(forecasts)
actual <- matrix(rep(true_y, n_models), ncol = n_models)

loss <- (actual - forecasts) ^ 2
best_models <- MCSprocedure(loss)
```

```
##
## ###########################################################################
## Superior Set Model created   :
##    Rank_M        v_M MCS_M Rank_R           v_R MCS_R        Loss
## m1      2 -1.1547319     1      2  7.920634e-05     1 0.009990946
## m2      1 -1.1549099     1      1 -7.920634e-05     1 0.009987804
```

```
## m3      3 -1.1535100     1      3  9.536588e-04     1 0.010025636
## m4      4  0.8646229     1      4  1.321971e+00     0 0.062449207
## m5      6  0.8661075     1      6  1.323123e+00     0 0.062489241
## m6      7  0.8675828     1      7  1.323871e+00     0 0.062524639
## m7      5  0.8652916     1      5  1.322372e+00     0 0.062465114
## p-value  :
## [1] 1
##
## ##########################################################################
```

- Квадратичная зависимость времени работы от количества моделей

Если увеличить количество моделей в 10 раз, то время работы возрастёт в 100 раз :)

Чтобы уменьшить время работы можно использовать несколько ядер процессора:

```
library("parallel")
```

Сначала узнаем, сколько ядер у доступного нам процессора

```
detectCores()
```

```
## [1] 8
```

Обычно оптимальная производительность достигается, если задействовано ядер чуть меньше, чем имеется в наличии. Возьмем 6 ядер для данного опыта!

```
cluster <- makeCluster(6)
best_models <- MCSprocedure(Loss_mini, verbose = FALSE, cl = cluster)
```

```
## Warning: 'memory.limit()' is Windows-specific

## Warning: 'memory.limit()' is Windows-specific

## Warning: 'memory.limit()' is Windows-specific
```

```
stopCluster(cluster)

print(best_models)
```

```
##
## ------------------------------------------
## -          Superior Set of Models         -
## ------------------------------------------
##                  Rank_M         v_M MCS_M Rank_R        v_R MCS_R
## gjrGARCH-snorm       1 -1.90662215     1      1 -0.5702230     1
## gjrGARCH-sstd        3 -0.61977052     1      3  0.8518862     1
## gjrGARCH-sged        5  0.16437187     1      5  1.3685004     1
## gjrGARCH-jsu         4 -0.01783499     1      4  1.2455776     1
## gjrGARCH-ghyp        6  0.17603441     1      6  1.3719314     1
## apARCH-norm          2 -1.02272715     1      2  0.5693786     1
```

```
## apARCH-std           8  1.09265494     1      8  2.0097864     1
## apARCH-ged           9  1.41204732     1      9  2.2177794     0
## apARCH-snorm         7  0.74386764     1      7  1.7745224     1
##                           Loss
## gjrGARCH-snorm 0.0004316483
## gjrGARCH-sstd  0.0004326200
## gjrGARCH-sged  0.0004332168
## gjrGARCH-jsu   0.0004330774
## gjrGARCH-ghyp  0.0004332255
## apARCH-norm    0.0004322969
## apARCH-std     0.0004339199
## apARCH-ged     0.0004341585
## apARCH-snorm   0.0004336560
##
## Details
## ------------------------------------------
##
## Number of eliminated models  :   2
## Statistic    :   Tmax
## Elapsed Time :   Time difference of 9.646628 secs
```

В конце надо обязательно остановить кластер :)

Почиташки:

Hansen, Lunde, Nason - 2011 Здесь T_R, T_{max}

Hansen, Lunde, Nason - 2003 Здесь T_{max}, T_{SQ}

Bernardi, Catania Виньетка пакета MCS.