2015 Discover Cup University Modeling Contest

****

**SIMPLE PREDICTION MODEL**

**Designed by Liu Chang**

Contents

Key Personnel 3

1. Abstract 4

2. Model Methodology 5

3. Model Development Data 7

4. Tools, Codes and Appendices 9

5. Model Dependent & Independent Variable definitions 16

6. Model Limitations and Weaknesses 17

# Key Personnel

Developer : Liu Chang

School Name : University of Science and Technology of China

Expected Graduation Date : Apr-2018

Expected Degree : Master

Major : Software Engineering

Email : [sa615422@mail.ustc.edu.cn](mailto:sa615422@mail.ustc.edu.cn)

Cell Phone : 188 9677 8499

# Abstract

It is easy to find a model to predict the future data, but calculating the unkown records accurately is difficult. It is hard for my laptop to collect a large number of items in the **data.txt**, so I designed a simple structure to store some fileds and write them into other file named **MoreSimple.db.**

I found that two factors affect the final results though analyzing the **data.txt**, the one is the weekday and the other is the day of a month. But rule of thumb, the holiday should be one of them as well. Relations between CallVolume and the holiday couldn’t be found beyond analyzed, I just provided a interface based on the holiday, but not make it true.

In this Simple Prediction Model, the future result (callVolume, handlingTime ) of each day depends on average result on a daily base, the weight of every weekday and the weight every day of a month. It will be introduced in detail later.

# Model Methodology

This prediction model mainly depends on three formulas:

**For the value of CallVolume:**

***averageCallVolume = totalCallVolume / allDays***

***weightCallVolumeDay[i] = totalCallVolumeDay[i] / day[i]***

***weightCallVolumeWeekDay[i] = totalCallVolumeWeekDay[i] / weekday[i]***

***callVolume[i] = averageCallVolume \* weightCallVolumeWeekDay[i] \* weightCallVolumeDay[i]***

averageCallVolume: the average of total callVolume.

totalCallVolumeWeekDay[i]: the total callVolume of every weekday, such as the total of 48 Sunday’s callVolume.

totalCallVolumeDay[i]: the total callVolume of the first i day of these months, for example, there are 11 3th days from JAN to NOV.

day[i]: the total days of the first i day of these months.

callVolume[i] : the callVolume of the first i day of a month

Weekday[i]: the number of specified weekday, for example, there are 48 Sundays before Nov.1999.

**For the value of HandlingTime:**

It is similar to the callVolume, follow above formula, just replace callVolume with handlingTime

**For the value of callVolumeHalfHour:**

It needs to change the formula a little.

***CallVolumeHalfHour[i][k] = averageCallVolume[k] \* weightCallVolumeWeekDay[k][weekday] \* weightCallVolumeDay[i]***

CallVolumeHalfHour[i][k]: the first i day of DEC, the first k half hour in a day. For example, callVolumeHalfHour[1][3] means the callVolume between 1:00 and 1:30 in 1st December.

# Model Development Data

There are about four hundred thousand records in the data.txt, and I wrote a program named extractData.c to collect some useful messages. A structure needed to store these messages and to write them into **MoreSimple.db.**

Struct HalfHour

{

char time[10]; // current time, such as “00:30:00”

long volume; // the volume of half hour

int index; // the first index half hour in the day

};

Struct DataTotal

{

int weekday; // what day is it today ?

int holiday;

struct HalfHour halfHour[48];

long callVolume; // the callVolume of current day

long handlingTime;

};

It took me about about two hours to extract these structures and to write them into **MoreSimple.db**. After that, when calculating the final results, I can ignore **data.txt** and just read datas from **MoreSimple.db** , this process can be very fast.

Each item could be identified by the value of index, for example, it means Call\_date when index = 6, IRV\_TIME when index = 9 and set a flag when states have changed. I will show you one part of my **extractData.c** codes.

……..

// the value of result comes from splitting string which is read from **MoreSimple.db**

If( ( index == 6 ) && strcmp(result, date ) == 0 )

{

callVolume++;

flag = 1;

}

// tmp and current\_time mean current half hour of this day.

if( ( flag == 1) && ( index == 7) )

{

if( compare(result, tmp) >= 0 && compare(result, current\_time) < 0 )

{

check = 1;

callHalfHour++;

}

}

// index = 9 means IRV\_TIME

if( ( flag == 1 ) && ( index == 9 ) )

{

if( atoi(result) < 0 ) // this record is invalid ( IRV\_TIME < 0 )

{

callVolume--;

flag = 2;

if( check == 1 )

callHalfHour--;

}

}

……

For **analysisData.c,** reading these messages into structure arrays. According to these arrays, we can calculate the final results.

# Tools, Codes and Appendices

OS : MAC OS X Yosemite

Code editor : Vim

Programming language : C

I calculated some specific variables and found out some factors which can affect the final result. They will be revealed below as a reference.

-------------------------------------------------------------------------------------------------

1999.1.11 && 1999.1.12

1999.5.21 && 1999.9.11 callVolume = 0;

totalCallVolume = 401053 totalHandlingTime = 60339351

-------------------------------------------------------------------------------------------------

**The rank of totalCallVolume during a day:**

1999-5-23 callVolume = 3022 handlingTime = 271449 weekday = SUN

1999-7-4 callVolume = 2586 handlingTime = 262614 weekday = SUN

1999-8-1 callVolume = 2372 handlingTime = 292596 weekday = SUN

1999-8-2 callVolume = 2361 handlingTime = 340814 weekday = MON

1999-8-3 callVolume = 2344 handlingTime = 278714 weekday = TUE

1999-3-11 callVolume = 2250 handlingTime = 245333 weekday = THU

1999-11-2 callVolume = 2223 handlingTime = 330603 weekday = TUE

**The rank of totalHandlingTime during a day:**

1999-11-15 handlingTime = 359380 callVolume = 2016 weekday = MON

1999-8-2 handlingTime = 340814 callVolume = 2361 weekday = MON

1999-11-2 handlingTime = 330603 callVolume = 2223 weekday = TUE

1999-5-2 handlingTime = 321079 callVolume = 2044 weekday = SUN

1999-1-3 handlingTime = 319282 callVolume = 2110 weekday = SUN

1999-11-1 handlingTime = 317123 callVolume = 2021 weekday = MON

1999-9-13 handlingTime = 315908 callVolume = 1844 weekday = MON

-------------------------------------------------------------------------------------------------

**Some datas about the weekday:**

SUN(48 days ) totalCallVolume = 79637 totalHandlingTime = 11576016

averageCallVolume = 1659.1041 averageHandlingTime = 21167

MON(47 days ) totalCallVolume = 72421 totalHandlingTime = 11146091

averageCallVolume = 1540.8723 averageHandlingTime = 237150.8723

TUE(47 days ) totalCallVolume = 74007 totalHandlingTime = 11207633

averageCallVolume = 1574.6170 averageHandlingTime = 238460.2766

WED(47 days ) totalCallVolume = 69034 totalHandlingTime = 10681335

averageCallVolume = 1468.8085 averageHandlingTime = 227262.4468

THU(47 days ) totalCallVolume = 70416 totalHandlingTime = 10898069

averageCallVolume = 1498.2128 averageHandlingTime = 231873.8085

FRI( 47 days ) totalCallVolume = 25810 totalHandlingTime = 3609453

averageCallVolume = 549.1489 averageHandlingTime = 76796.8723

SAT(47 days ) totalCallVolume = 9685 totalHandlingTime = 1220754

averageCallVolume = 206.0638 averageHandlingTime = 25973.4894

-------------------------------------------------------------------------------------------------

**Some datas about the month:**

JAN：( 31 – 2 = 29 days ) // the callVolume of 12nd and 13rd is empty.

callVolume = 31566 totalHandlingTime = 4960803

averageCallVolume = 1088.4828 averageHandlingTime = 171062.1719

FEB：( 28 days )

callVolume = 33319 totalHandlingTime = 4787215

averageCallVolume = 1189.9642 averageHandlingTime = 170971.9688

MAR：( 31 days )

callVolume = 38774 totalHandlingTime = 5169873

averageCallVolume = 1250.7742 averageHandlingTime = 166770.0938

APR：( 30 days )

callVolume = 31999 totalHandlingTime = 5016831

averageCallVolume = 1066.6333 averageHandlingTime = 167227.7031

MAY：( 31 – 1 = 30 days )

callVolume = 39461 totalHandlingTime = 5960121

averageCallVolume = 1315.3667 averageHandlingTime = 198670.7031

JUN：( 30 days )

callVolume = 37908 totalHandlingTime = 5888159

averageCallVolume = 1263.6000 averageHandlingTime = 196271.9688

JUL：( 31 days )

callVolume = 38976 totalHandlingTime = 5429026

averageCallVolume = 1257.2903 averageHandlingTime = 175129.8750

AGU：( 31 days )

callVolume = 42059 totalHandlingTime = 5788999

averageCallVolume = 1356.7412 averageHandlingTime = 186741.9063

SEP：( 30 – 1 = 29 days )

callVolume = 31328 totalHandlingTime = 5185386

averageCallVolume = 1080.2759 averageHandlingTime = 178806.4063

OCT：( 31 days )

callVolume = 34621 totalHandlingTime = 5558078

averageCallVolume = 1116.8064 averageHandlingTime = 179292.8437

NOV：( 30 days )

callVolume = 40999 totalHandlingTime = 6594860

averageCallVolume = 1336.6330 averageHandlingTime = 219828.6719

-------------------------------------------------------------------------------------------------

**I will give some codes about this part.**

This part reads message from MoreSimple.db and writes them into dataTotal arrays.

#if 1

for( Month = 1; Month <= 11; Month++ )

{

for( Day = 1; Day <= 31; Day++ )

{

if( Month == 2 && Day == 29 ) // Only 28 days in Feb, 1999

break;

if( ( Month == 4 || Month == 6 || Month == 9 || Month == 11 ) && Day == 31 ) // only 30 days

break;

if( fread(&dataTotal[Month-1][Day-1], sizeof(struct DataTotal), 1, fp\_data) <=0 )

{

fprintf(stderr, "fread error!\n");

exit(EXIT\_FAILURE);

}

fflush(fp\_data);

}

}

-------------------------------------------------------------------------------------------------

This part calculates some important variables:

for( Month = 1; Month <= 11; Month++ )

{

for( Day = 1; Day <= 31; Day++ )

{

if( ( Month == 1 && Day == 11 ) || ( Month == 1 && Day == 12 )) // the value is empty

continue;

if( Month == 2 && Day == 29 ) // skip

break;

if( ( Month == 4 || Month == 6 || Month == 9 || Month == 11 ) && Day == 31 ) // skip

break;

if( Month == 5 && Day == 21 ) // the value is empty this day

continue;

if( Month == 9 && Day == 11 ) // the value is empty this day

continue;

for( k = 0; k < 48; k++ ) // there are 48 half hours in a day

{

totalCallVolumeHalfHour[k] += dataTotal[Month-1][Day-1].halfHour[k].volume;

}

day[Day-1]++; // the total of the first i day in these months

callVolumeDay[Day-1] += dataTotal[Month-1][Day-1].callVolume;

handlingTimeDay[Day-1] += dataTotal[Month-1][Day-1].handlingTime;

#if 1

switch( dataTotal[Month-1][Day-1].weekday )

{

case 0: // Sunday

{

for( k = 0; k < 48; k++ )

{

halfHourCallVolume[k][0] += dataTotal[Month-1][Day-1].halfHour[k].volume;

}

weekDay[0]++; // the total of Sunday

weekCallVolume[0] += dataTotal[Month-1][Day-1].callVolume;

weekHandlingTime[0] += dataTotal[Month-1][Day-1].handlingTime;

break;

}

case 1: // Monday

{

for( k = 0; k < 48; k++ )

{

halfHourCallVolume[k][1] += dataTotal[Month-1][Day-1].halfHour[k].volume;

}

weekDay[1]++;

weekCallVolume[1] += dataTotal[Month-1][Day-1].callVolume;

weekHandlingTime[1] += dataTotal[Month-1][Day-1].handlingTime;

break;

}

case 2: // Tuesday

{

for( k = 0; k < 48; k++ )

{

halfHourCallVolume[k][2] += dataTotal[Month-1][Day-1].halfHour[k].volume;

}

weekDay[2]++;

weekCallVolume[2] += dataTotal[Month-1][Day-1].callVolume;

weekHandlingTime[2] += dataTotal[Month-1][Day-1].handlingTime;

break;

}

case 3: // Wedsday

{

for( k = 0; k < 48; k++ )

{

halfHourCallVolume[k][3] += dataTotal[Month-1][Day-1].halfHour[k].volume;

}

weekDay[3]++;

weekCallVolume[3] += dataTotal[Month-1][Day-1].callVolume;

weekHandlingTime[3] += dataTotal[Month-1][Day-1].handlingTime;

break;

}

case 4: // Thursday

{

for( k = 0; k < 48; k++ )

{

halfHourCallVolume[k][4] += dataTotal[Month-1][Day-1].halfHour[k].volume;

}

weekDay[4]++;

weekCallVolume[4] += dataTotal[Month-1][Day-1].callVolume;

weekHandlingTime[4] += dataTotal[Month-1][Day-1].handlingTime;

break;

}

case 5: // Friday

{

for( k = 0; k < 48; k++ )

{

halfHourCallVolume[k][5] += dataTotal[Month-1][Day-1].halfHour[k].volume;

}

weekDay[5]++;

weekCallVolume[5] += dataTotal[Month-1][Day-1].callVolume;

weekHandlingTime[5] += dataTotal[Month-1][Day-1].handlingTime;

break;

}

case 6: // Saturday

{

for( k = 0; k < 48; k++ )

{

halfHourCallVolume[k][6] += dataTotal[Month-1][Day-1].halfHour[k].volume;

}

weekDay[6]++;

weekCallVolume[6] += dataTotal[Month-1][Day-1].callVolume;

weekHandlingTime[6] += dataTotal[Month-1][Day-1].handlingTime;

break;

}

default:

break;

}

allDays++; // the total days in these months

totalCallVolume += dataTotal[Month-1][Day-1].callVolume;

totalHandlingTime += dataTotal[Month-1][Day-1].handlingTime;

#endif

}

}

#endif

-------------------------------------------------------------------------------------------------

This parts begins to calculate the weight.

#if 1

averageCallVolume = (totalCallVolume / (float)(allDays));

averageHandlingTime = (totalHandlingTime / (float)allDays);

for( k = 0; k < 7; k++ )

{

weightCallVolumeWeekDay[k] = ( weekCallVolume[k] / (float)weekDay[k] ) / averageCallVolume;

weightHandlingTimeWeekDay[k] = ( weekHandlingTime[k] / (float)weekDay[k] ) / averageHandlingTime;

}

for( k = 0; k < 31; k++ )

{

weightCallVolumeDay[k] = ( callVolumeDay[k] / (float)day[k] ) / averageCallVolume;

weightHandlingTimeDay[k] = ( handlingTimeDay[k] / (float)day[k] ) / averageHandlingTime;

}

#if 1

for( k = 0; k < 48; k++ )

{

averageCallVolumeHalfHour[k] = totalCallVolumeHalfHour[k] / (float)(allDays);

}

#if 1

for( k = 0; k < 48; k++ )

{

for( i = 0; i < 7; i++ )

{

weightCallVolumeHalfHour[k][i] =( ( halfHourCallVolume[k][i] / (float)weekDay[i] ) ) / averageCallVolumeHalfHour[k];

}

}

#endif

-------------------------------------------------------------------------------------------------

This part begins to calculate the final results and save them in arrys.

#if 1

for( Month = 12; Month <= 12; Month++ )

{

for( Day = 1; Day <= 31; Day++ )

{

strcpy( current\_time , "00:00:00" );

weekday = getWeek(1999, Month, Day);

dataTotal[Month-1][Day-1].weekday = weekday;

dataTotal[Month-1][Day-1].holiday = getHoliday(1999, Month, Day); // this is just a interface

#if 1

for( k = 0; k < 48; k++ )

{

dataTotal[Month-1][Day-1].halfHour[k].index = k;

strcpy( dataTotal[Month-1][Day-1].halfHour[k].time, current\_time );

addHalfHour( current\_time ); // such as “00:00:00” 🡪 “00:30:00”

dataTotal[Month-1][Day-1].halfHour[k].volume = averageCallVolumeHalfHour[k] \* weightCallVolumeHalfHour[k][weekday] \* weightCallVolumeDay[Day-1] \* getHoliday(1999, Month, Day); // calculate the value of every half hour in a day.

}

#endif

dataTotal[Month-1][Day-1].callVolume = averageCallVolume \* weightCallVolumeWeekDay[weekday] \* weightCallVolumeDay[Day-1] \* getHoliday(1999, Month, Day);

dataTotal[Month-1][Day-1].handlingTime = averageHandlingTime \* weightHandlingTimeWeekDay[weekday] \* weightHandlingTimeDay[Day-1] \* getHoliday(1999, Month, Day);

}

}

-------------------------------------------------------------------------------------------------

# Model Dependent & Independent Variable definitions

char current\_time[10]; // current half hour of a day

int weekday = 0; // what day is it today

int weekDay[7]; // weekday[i]: the total value of the first i weekday.

int day[31]; // the total of the first i day in these months

long weekCallVolume[7]; // the total callVolume of the first i weekday

long weekHandlingTime[7]; // the total handlingTime of the first i weekday

long halfHourCallVolume[48][7]; //total callVolume of the first i half hour in the i weekday

long callVolumeDay[31]; // total callVolume of the first i day in these months.

long handlingTimeDay[31];

long totalCallVolume = 0; // total callVolume in all days

long totalHandlingTime = 0;

long totalCallVolumeHalfHour[48]; // total callVolume of the first i half hour in all days

int allDays = 0;

float weightCallVolumeWeekDay[7]; // the weight about weekday

float weightHandlingTimeWeekDay[7];

float weightCallVolumeHalfHour[48][7]; // the weight about first i half hour and weekday

float weightCallVolumeDay[31];

float weightHandlingTimeDay[31];

float averageCallVolume = 0.0;

float averageHandlingTime = 0.0;

float averageCallVolumeHalfHour[48]; // the average of every half hour

struct DataTotal dataTotal[MONTH][DAY]; // this array is to store these messages.

# Model Limitations and Weaknesses

This model is very simple and incomeplete....I also try some other models such as GM(1,1) and Moving-Average, but these given datas are not sutiable well....

In this progress, I try my best to complete this model on my own and never consider giving it up. I really do a lot for it and learn a lot from it. Thanks for the Discover Contest! Thank you!