

VONeural Project

Confusion Matrix Library

Doc. : confusion_matrix_library_VONEURAL-SPE-NA-0001-Rel1.0.odt

Issue: 1.0

Date: 07/07/2007

What	Who	When	Signatures
Prepared by	Stefano Cavuoti	07/07/2007	
Approved by			
Released by			

Revision Matrix

Issue	Author	Date	Section/Paragraph Affected	Reason/Initiation/ Documents/Remarks
1.0	Stefano Cavuoti	07/07/2007	All	First Issue

INDEX

- 1. Reference & Applicable Documents.....4
- 2. Abbreviations & Acronyms.....5
- 3. Introduction.....6
- 4. Functions.....6
 - 4.1. alloc_confusion_matrix and alloc_confusion_matrix_advanced.....6
 - 4.2. update_confusion_matrix and update_confusion_matrix_advanced6
 - 4.3. print_confusion_matrix and print_confusion_matrix_advanced6
 - 4.4. destroy_conf_mat and destroy_conf_mat_advanced6
- 5. Appendix7
 - A. Source Code.....7
 - B. Exemple.....10

TABLE INDEX

- Tab. 1 – Reference Documents.....4
- Tab. 2 – Abbreviations and acronyms.....5

FIGURE INDEX

1. Reference & Applicable Documents

REF. ID	TITLE	DATE	AUTHOR	COMMENTS
REF. 1	statement of work VONEURAL-SOW-NA-00001-Rel1.1.doc	06/06/2007	M. Brescia	Draft

Tab. 1 – Reference Documents

2. Abbreviations & Acronyms

A & A	Meaning

Tab. 2 – Abbreviations and acronyms

3. Introduction

This package of functions serves to produce simple diagnostic instruments that aim to estimate the efficiency of the produced net. The first and most important is the confusion matrix. It is a matrix with the values of target vector and the output values produced from the net respectively on its rows and columns. In addition this package allows to calculate the success rate, i.e. the percentage of objects correctly classified from the net. The number of "bit fault" that is objects badly classifies and the percentage of correctly classified objects for each class.

4. Functions

There's two versions of each function, normal one and advanced one, the difference is in format of the target and output vector, normal version is suitable for vectors of one column and "num output" class, advanced version instead is suitable for vectors of "num output" column each one with 2 class(0 and 1) Matrix produced has got on the raw target vector the first row is class 0 second class 1, first column is the class 0 of output vector second column is class 1 and so on, for multi column vector there's a simple conversion binary to decimal, so we have in the case of a two column vector 00 on the first row(column for the output obviously) 01 on the second 10 on the third and 11 on the forth.

4.1. `alloc_confusion_matrix` and `alloc_confusion_matrix_advanced`

`alloc_confusion_matrix` and `alloc_confusion_matrix_advanced` init a N x N matrix where N is the number of class needed.

Input: Number of class/Number of column according the choose of the function

Output: An empty N*N matrix where N is the number of class needed.

4.2. `update_confusion_matrix` and `update_confusion_matrix_advanced`

`update_confusion_matrix` and `update_confusion_matrix_advanced` taked a target vector and one output vector add 1 to the right element of the matrix.

Input: Confusion Matrix
Target Vector
Output Vector
Number of class/Number of column according the choose of the function

Output: A modified Confusion Matrix

4.3. `print_confusion_matrix` and `print_confusion_matrix_advanced`

`print_confusion_matrix` and `print_confusion_matrix_advanced` calculate success rate, bit fault, success rate of the classes, and print that on the output file.

Input: Number of class/Number of column according the choose of the function
Confusion Matrix
Output File Name

Output: Output File

4.4. `destroy_conf_mat` and `destroy_conf_mat_advanced`

`destroy_conf_mat` and `destroy_conf_mat_advanced` free memory cells used by confusion matrix.

Input: Confusion Matrix,
Number of class/Number of column according the choose of the function

Output: none

5. Appendix

A. Source Code

```
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
#include<string.h> /* _advanced function are multicolumn ones */
int ** alloc_confusion_matrix(int num_output) {
    int ** confusion_matrix = calloc( num_output , sizeof(int *));
    int i;
    for(i = 0; i != num_output; i++)
    {
        confusion_matrix[i] = calloc(num_output , sizeof(int));
    }
    return confusion_matrix;
}
int update_confusion_matrix(int ** confusion_matrix, float target,
float output, int num_output) {
    int i, j;
    i=0;
    j=0;
    for(i=0; i != num_output && abs(target-i)>0.5; i++);
    for(j=0; j != num_output && abs(output-j)>0.5; j++);
    confusion_matrix[i][j]++;
    return 0;
}
int print_confusion_matrix (int num_output, int **
confusion_matrix, char * log_file) {
    int i, j;
    int total = 0;
    int success = 0;
    int * successclass = calloc( num_output , sizeof(int));
    int * totalbit = calloc(num_output , sizeof(int));
    FILE * stream = fopen(log_file, "w");
    for(i = 0; i != (num_output); i++)
    {
        success +=confusion_matrix[i][i];
        successclass[i]=confusion_matrix[i][i];
        totalbit[i]=0;
        for( j = 0; j != num_output; j++)
        {
            fprintf(stream, "%lu" , confusion_matrix[i][j]);
            fprintf(stream, "\t");
            total+= confusion_matrix[i][j];
            totalbit[i]+= confusion_matrix[i][j];
        }
        fprintf(stream, "\n");
    }
    fprintf(stream, "\nSuccess rate is ");
    float succrat = 100*success/total;
    fprintf(stream, "%f" ,succrat);
    fprintf(stream, "% \n \n");
    int bitfault = total - success;
```

```

fprintf(stream, "%hu", bitfault);
fprintf(stream, " Bits are fault\n\n");
float * successpercent = calloc(num_output , sizeof(float));
int k = 0;
for(k = 0; k != num_output; k++)
{
    fprintf(stream, "Success rate for class ");
    fprintf(stream, "%hu" , k);
    fprintf(stream, " is: \t");
    if (totalbit[k]!=0)
    {
        successpercent[k]=100*successclass[k]/totalbit[k];
        fprintf(stream, "%f" , successpercent[k]);
        fprintf(stream, "% \n");
    }
    else
    {
        fprintf(stream, "class not present in the dataset\n");
    }
}
free(successclass);
free(totalbit);
free(successpercent);
fclose(stream);
}

int destroy_conf_mat(int ** confusion_matrix, int num_output) {
    int i;
    for(i = 0; i != num_output; i++)
    {
        free(confusion_matrix[i]);
    }
    free(confusion_matrix);
}

int ** alloc_confusion_matrix_advanced(int num_output) {
    int ** confusion_matrix = calloc( (int)pow(2, num_output) , sizeof(int *));
    int i;
    for(i = 0; i != pow(2, num_output); i++)
        confusion_matrix[i] = calloc((int)pow(2, num_output) , sizeof(int));
    if (confusion_matrix == NULL)
    {
        printf("Memory not allocated in matrix");
    }
    else
        return confusion_matrix;
}

int update_confusion_matrix_advanced(int ** confusion_matrix,
float * target, float * output, int num_output) {
    int i, indicei , indicej;
    i=0;
    indicei=0;
    indicej=0;
    while (i != num_output)
    {
        if (target[i]<0.5)
        {
            target[i]=0;
        }
        else

```

```

    {
        target[i]=1;
    }
    indicei+= target[i]*(pow(2 , (num_output-i-1)));
    if (output[i]<0.5)
    {
        output[i]=0;
    }
    else
    {
        output[i]=1;
    }
    indicej+= output[i]*(pow(2 , (num_output-i-1)));
    i++;
}
confusion_matrix[indicei][indicej]++ ;
return 0;
} int print_confusion_matrix_advanced (int num_output, int **
confusion_matrix, char * log_file) {
    int i, j;
    int total = 0;
    int success = 0;
    int * successclass = calloc(pow(2, num_output) , sizeof(int));
    int * totalbit = calloc(pow(2, num_output) , sizeof(int));
    FILE * stream = fopen(log_file, "w");
    for(i = 0; i != pow(2, num_output); i++)
    {
        success +=confusion_matrix[i][i];
        successclass[i]=confusion_matrix[i][i];
        totalbit[i]=0;
        for(j = 0; j != pow(2, num_output); j++)
        {
            fprintf(stream, "%lu" , confusion_matrix[i] [j]);
            fprintf(stream, "\t");
            total+= confusion_matrix[i][j];
            totalbit[i]+= confusion_matrix[i][j];
        }
        fprintf(stream, "\n");
    }

    fprintf(stream, "\nSuccess rate is ");
    float succrat = success/total;
    fprintf(stream, "%f" ,succrat);
    fprintf(stream, "\n \n");
    int bitfault = total - success;
    fprintf(stream, "%hu", bitfault);
    fprintf(stream, " Bits are fault\n\n");
    float * successpercent = calloc(pow(2, num_output) , sizeof(float));
    int k = 0;
    for(k = 0; k != pow(2, num_output); k++)
    {
        printf("\n");
        /* char foo(int class)
        {
            char b = "";
            while (class > 0)
            {
                int    j = i & 1;

```

```

        char sid;
        sid=sprintf(sid, "%d" ,j);
        b=strcat( sid , b);
        class >= 1;
    }
    int z;
    for(z = 0; z != (num_output-strlen(b)); z++)
    {
        b= strcat( "0" , b);
    }
    return b;
}
char classbin = foo(k);
fprintf(stream, classbin); */
if (totalbit[k]!=0)
{
    fprintf(stream, "Success rate for class ");
    fprintf(stream, "%hu" , k);
    fprintf(stream, " is: \t");
    successpercent[k]=successclass[k]/totalbit[k];
    fprintf(stream, "%f" , successpercent[k]);
    fprintf(stream, "% \n");
}
else
{
    fprintf(stream, "class ");
    fprintf(stream, "%hu", k);
    fprintf(stream, " not present in the dataset\n");
}
}
free(successclass);
free(totalbit);
free(successpercent);
fclose(stream);
} int destroy_conf_mat_advanced(int ** confusion_matrix, int
num_output) {
    int i;
    for(i = 0; i != pow(2, num_output); i++)
    {
        free(confusion_matrix[i]);
    }
    free(confusion_matrix);
}

```

B. Exemple

```

int main() {
    char * log_file = "/home/cavuoti/Desktop/confusion.txt";
    int num_output = 3;

    int ** confusion_matrix = alloc_confusion_matrix(num_output);

    float target1[5];
    target1[1]=0;
    target1[2]=2;
    target1[3]=1;
    target1[4]=0;
    target1[0]=1;
    float output1[5];
}

```

```
output1[1]=0;
output1[2]=2.4;
output1[3]=1;
output1[4]=0;
output1[0]=0;
float target[5];
float output[5];
int k;
```

```
for (k=0; k!=5; k++)
```

```
{
    target[k]=1*target1[k];
    output[k]=1*output1[k];
```

```
    update_confusion_matrix(confusion_matrix, target[k], output[k], num_output);
}
```

```
print_confusion_matrix (num_output, confusion_matrix, log_file);
destroy_conf_mat(confusion_matrix, num_output);
```

__oOo__