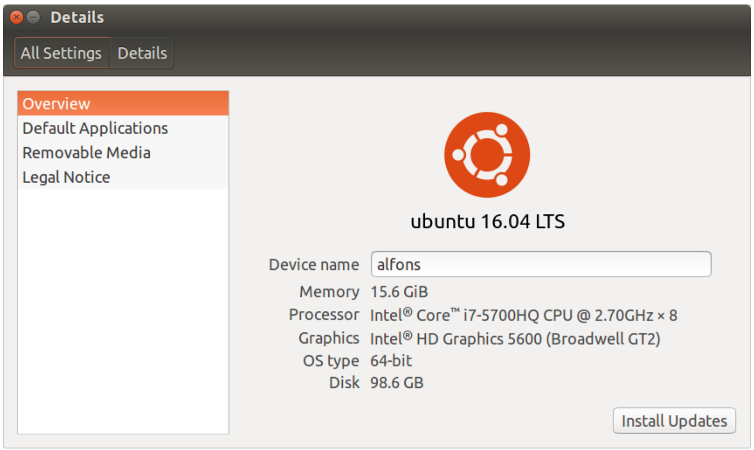
Introduction to Machine Learning Project 1

0416324 An-Fong Hwu ,Dept of Computer Science

1.Programming Environment



ubuntu 16.04 LTS

Device name alfons
Memory 15.6 GiB
Processor Intel® Core™ i7-5700HQ CPU @ 2.70GHz × 8
Graphics Intel® HD Graphics 5600 (Broadwell GT2)
OS type 64-bit
Disk 98.6 GB

Programming language used: C++ with -std=c++11 standard and library with #include <bits/stdc++.h>

2.What is Decision Tree and Random Forest?

A simple machine learning and training model for data prediction and analysis.

https://en.wikipedia.org/wiki/Decision_tree

https://en.wikipedia.org/wiki/Random_forest

3.How decision tree is built?

0.Store the data into the set of vector

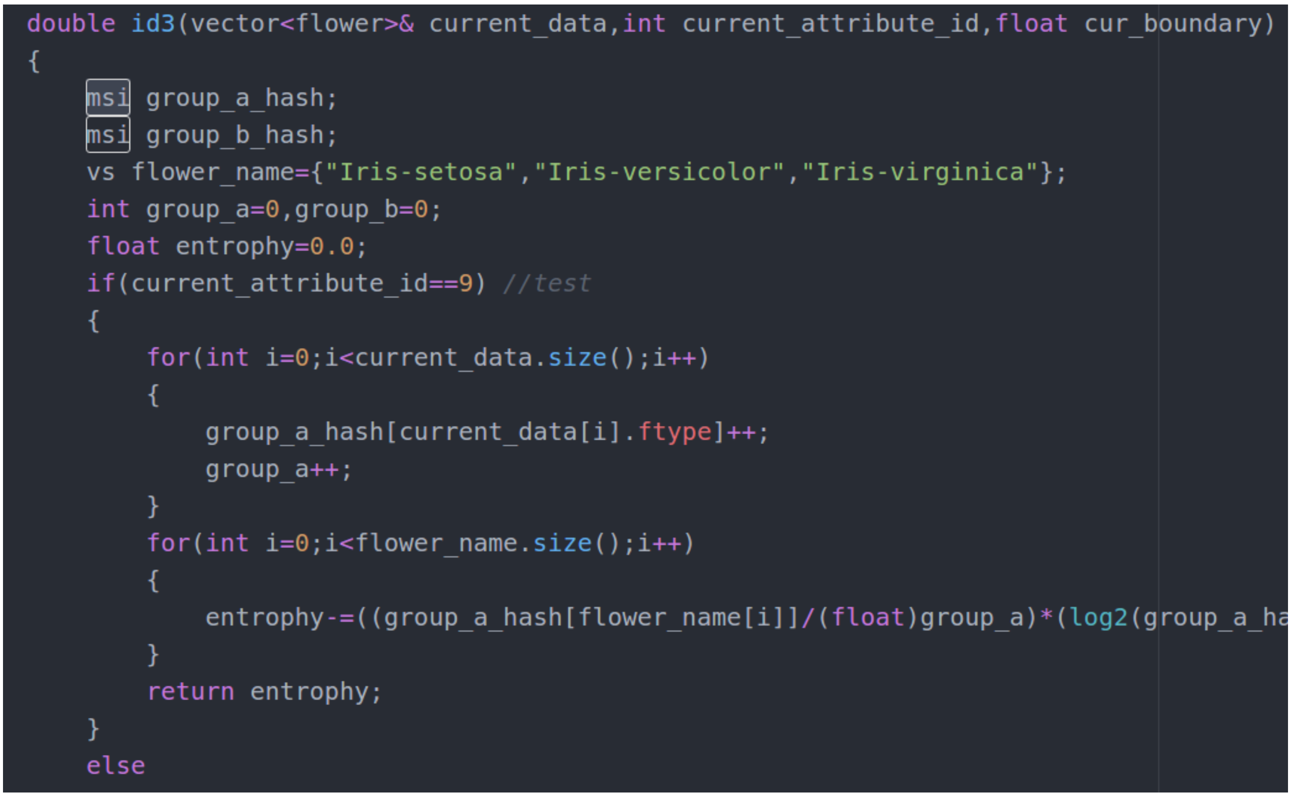
1.Sort according to the attribute (dose not matter which attribute will get the most information gain since all splitting and attribute will be calculated)

2.If different at index then we calculate at (or say split with (value[index]+value[index-1])/2)

3.Now the table has been split into 2 parts, then calculate ”each splitting point”according to the ID3 algorithm

By using the std::map, we will increase the convenience to make a statistics of dataset.

After calculating the position of splitting with LEAST ENTROPY , which means the ”chaos” of data is the LEAST, then we split at such position to reduce the data inconsistency.

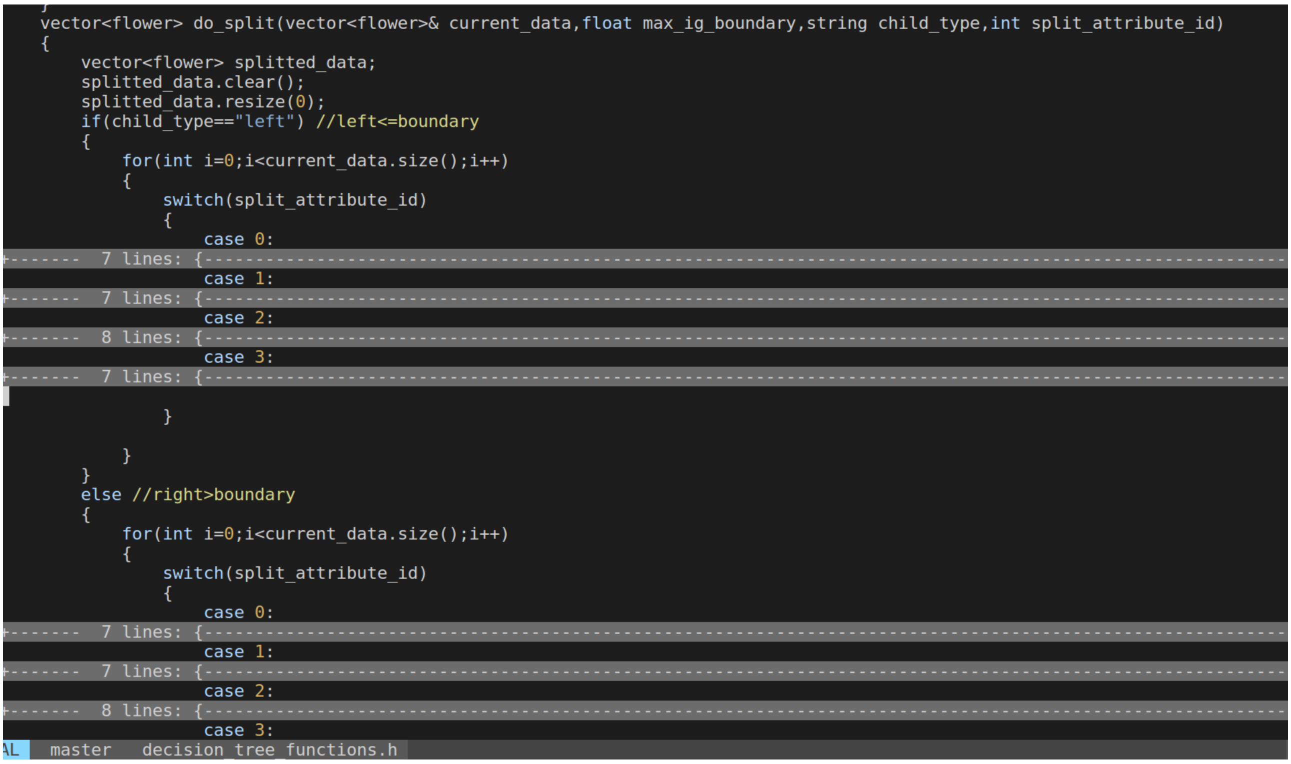


```
double id3(vector<flower>& current_data,int current_attribute_id,float cur_boundary)
{
    msi group_a_hash;
    msi group_b_hash;
    vs flower_name={"Iris-setosa","Iris-versicolor","Iris-virginica"};
    int group_a=0,group_b=0;
    float entrophy=0.0;
    if(current_attribute_id==9) //test
    {
        for(int i=0;i<current_data.size();i++)
        {
            group_a_hash[current_data[i].ftype]++;
            group_a++;
        }
        for(int i=0;i<flower_name.size();i++)
        {
            entrophy-=((group_a_hash[flower_name[i]]/(float)group_a)*(log2(group_a_ha
        }
        return entrophy;
    }
    else
```

4.We now have left_child and right_child. We split ,and new* left_child right child, connect them parent->newchild= something

6.take the needed data into leftchild which for example <180cm , then take all the person whose height <180cm into left child and vice versa for splitting the data set according to the current criterion.

Here is the code for putting the data in left child and right child



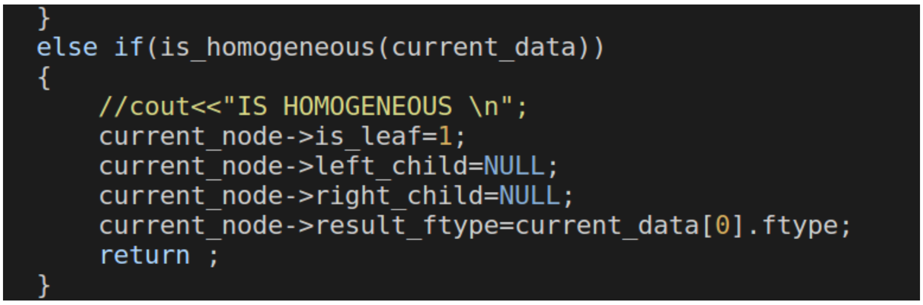
```
    }
    vector<flower> do_split(vector<flower>& current_data,float max_ig_boundary,string child_type,int split_attribute_id)
    {
        vector<flower> splitted_data;
        splitted_data.clear();
        splitted_data.resize(0);
        if(child_type=="left") //left<=boundary
        {
            for(int i=0;i<current_data.size();i++)
            {
                switch(split_attribute_id)
                {
                    case 0:
+-------  7 lines: {-----------------------------------------------------------------------------------------------
                    case 1:
+-------  7 lines: {-----------------------------------------------------------------------------------------------
                    case 2:
+-------  8 lines: {-----------------------------------------------------------------------------------------------
                    case 3:
+-------  7 lines: {-----------------------------------------------------------------------------------------------

                }

            }
        }
        else //right>boundary
        {
            for(int i=0;i<current_data.size();i++)
            {
                switch(split_attribute_id)
                {
                    case 0:
+-------  7 lines: {-----------------------------------------------------------------------------------------------
                    case 1:
+-------  7 lines: {-----------------------------------------------------------------------------------------------
                    case 2:
+-------  8 lines: {-----------------------------------------------------------------------------------------------
                    case 3:
AL   master   decision_tree_functions.h
```

5.If the node's data is homogeneous, stop (the node cannot be split even more).

6.the recursive algorithm is somehow like build_decision_tree(node* left_child) build_decision_tree(node* right_child) where the child is not null



```
}
else if(is_homogeneous(current_data))
{
    //cout<<"IS HOMOGENEOUS \n";
    current_node->is_leaf=1;
    current_node->left_child=NULL;
    current_node->right_child=NULL;
    current_node->result_ftype=current_data[0].ftype;
    return ;
}
```

Q:Which attribute to split first?

A:Does not matter, what matters is the boundary we split, the boundary has to bring us the most information gain

3.Extend to Random Forest

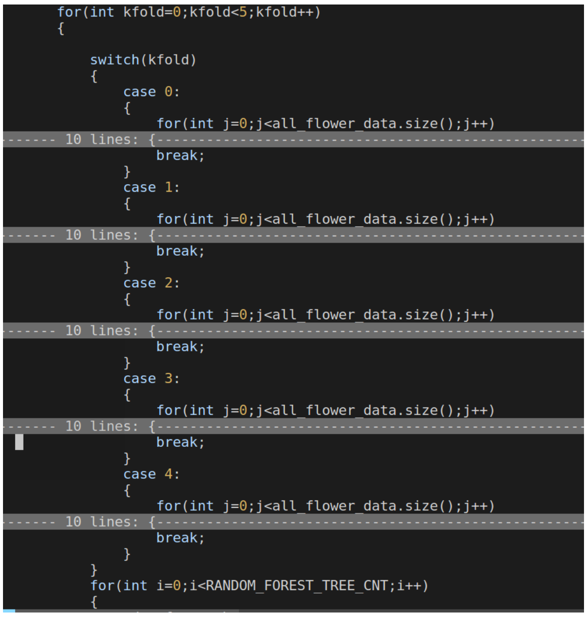
0.Build 5 decision tree where data picked from the data which require 120 training sets

1.Each tree contains 96 datasets, reason is that 120/24=5 and 120-24=96, just like the K-Fold interval for one decision tree, but now we have the subinterval for 5 trees and 24 as a count number for interval.

2.Traverse the forest, the highest vote for the predicted class is the.

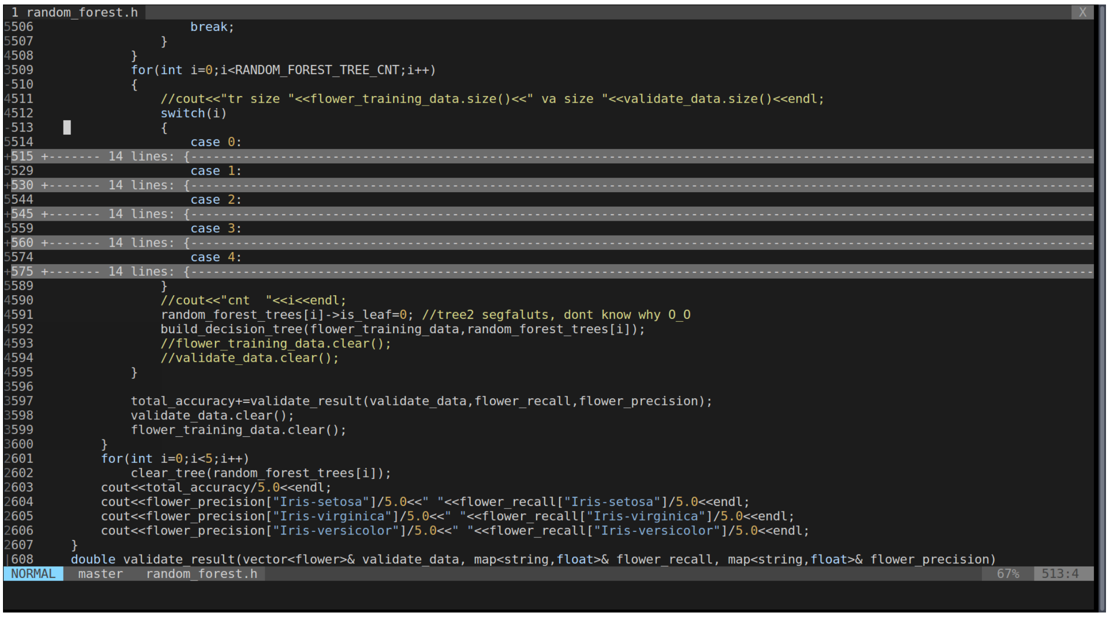
3.K Fold Cross validation still implementable.

4.Implement a set of root and using the same method to build the random forest



```
    for(int kfold=0;kfold<5;kfold++)
    {

        switch(kfold)
        {
            case 0:
            {
                for(int j=0;j<all_flower_data.size();j++)
------ 10 lines: {--------------------------------------------------
                break;
            }
            case 1:
            {
                for(int j=0;j<all_flower_data.size();j++)
------ 10 lines: {--------------------------------------------------
                break;
            }
            case 2:
            {
                for(int j=0;j<all_flower_data.size();j++)
------ 10 lines: {--------------------------------------------------
                break;
            }
            case 3:
            {
                for(int j=0;j<all_flower_data.size();j++)
------ 10 lines: {--------------------------------------------------
                break;
            }
            case 4:
            {
                for(int j=0;j<all_flower_data.size();j++)
------ 10 lines: {--------------------------------------------------
                break;
            }
        }
        for(int i=0;i<RANDOM_FOREST_TREE_CNT;i++)
        {
```

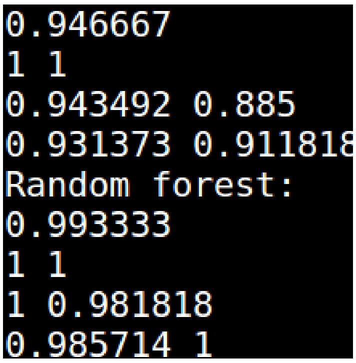


```
                        break;
                    }
                }
                for(int i=0;i<RANDOM_FOREST_TREE_CNT;i++)
                {
                    //cout<<"tr size "<<flower_training_data.size()<<" va size "<<validate_data.size()<<endl;
                    switch(i)
                    {
                        case 0:
+------- 14 lines: {-----------------------------------------------------------------------------------------------
                        case 1:
+------- 14 lines: {-----------------------------------------------------------------------------------------------
                        case 2:
+------- 14 lines: {-----------------------------------------------------------------------------------------------
                        case 3:
+------- 14 lines: {-----------------------------------------------------------------------------------------------
                        case 4:
+------- 14 lines: {-----------------------------------------------------------------------------------------------
                    }
                    //cout<<"cnt  "<<i<<endl;
                    random_forest_trees[i]->is_leaf=0; //tree2 segfaluts, dont know why O_O
                    build_decision_tree(flower_training_data,random_forest_trees[i]);
                    //flower_training_data.clear();
                    //validate_data.clear();
                }

                total_accuracy+=validate_result(validate_data,flower_recall,flower_precision);
                validate_data.clear();
                flower_training_data.clear();
            }
            for(int i=0;i<5;i++)
                clear_tree(random_forest_trees[i]);
            cout<<total_accuracy/5.0<<endl;
            cout<<flower_precision["Iris-setosa"]/5.0<<" "<<flower_recall["Iris-setosa"]/5.0<<endl;
            cout<<flower_precision["Iris-virginica"]/5.0<<" "<<flower_recall["Iris-virginica"]/5.0<<endl;
            cout<<flower_precision["Iris-versicolor"]/5.0<<" "<<flower_recall["Iris-versicolor"]/5.0<<endl;
        }
        double validate_result(vector<flower>& validate_data, map<string,float>& flower_recall, map<string,float>& flower_precision)
```

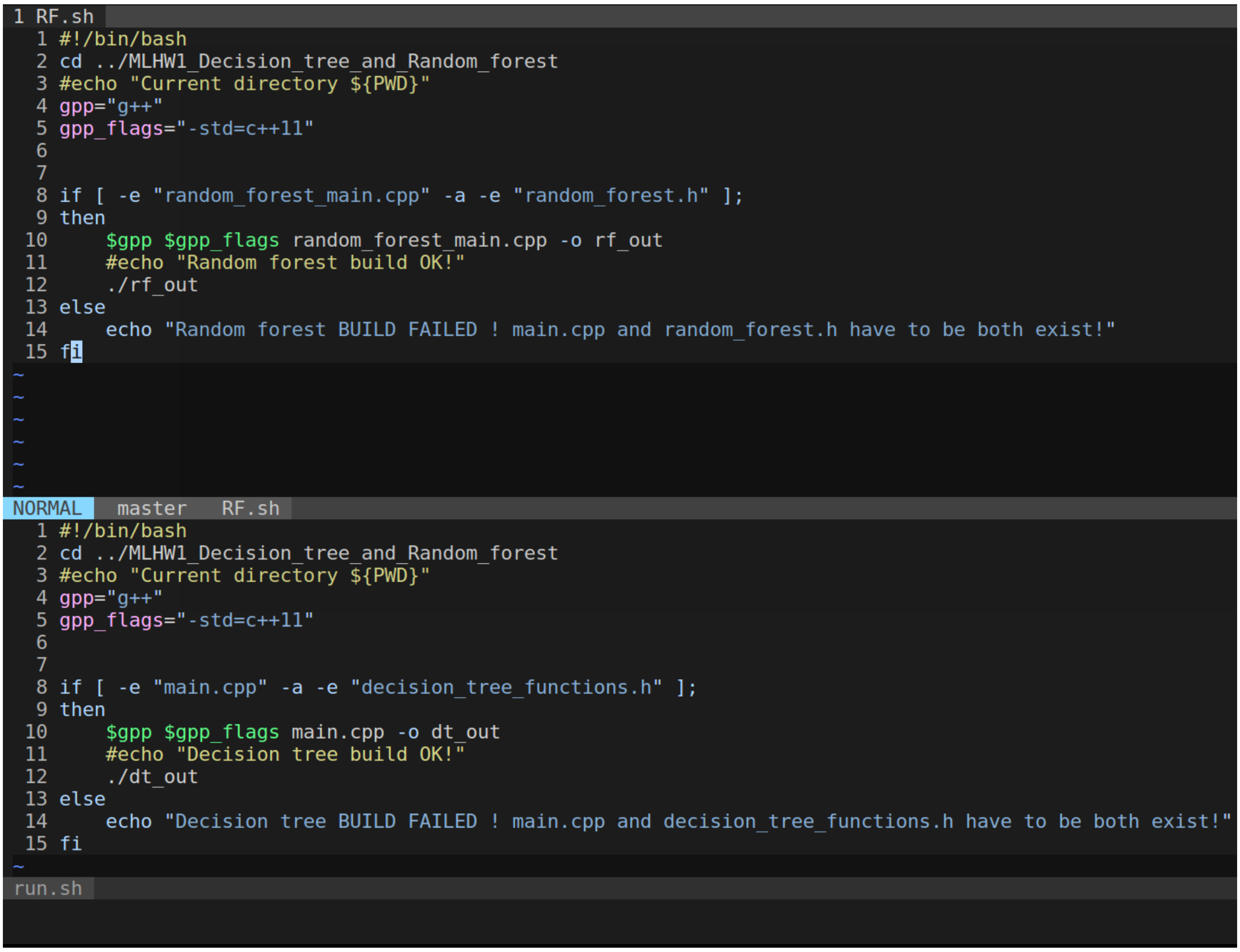
4.Validate the Result

There is a dramatically improved accuracy by using the Random Forest



```
0.946667
1 1
0.943492 0.885
0.931373 0.911818
Random forest:
0.993333
1 1
1 0.981818
0.985714 1
```

5.Shell script



```bash
#!/bin/bash
cd ../MLHW1_Decision_tree_and_Random_forest
#echo "Current directory ${PWD}"
gpp="g++"
gpp_flags="-std=c++11"


if [ -e "random_forest_main.cpp" -a -e "random_forest.h" ];
then
    $gpp $gpp_flags random_forest_main.cpp -o rf_out
    #echo "Random forest build OK!"
    ./rf_out
else
    echo "Random forest BUILD FAILED ! main.cpp and random_forest.h have to be both exist!"
fi
```

```bash
#!/bin/bash
cd ../MLHW1_Decision_tree_and_Random_forest
#echo "Current directory ${PWD}"
gpp="g++"
gpp_flags="-std=c++11"


if [ -e "main.cpp" -a -e "decision_tree_functions.h" ];
then
    $gpp $gpp_flags main.cpp -o dt_out
    #echo "Decision tree build OK!"
    ./dt_out
else
    echo "Decision tree BUILD FAILED ! main.cpp and decision_tree_functions.h have to be both exist!"
fi
```

cd the relative path first ,setting the compiling flag, checking the availability of file then execute.