Parallelized Random Forests Learning

CSE 633 Course Project

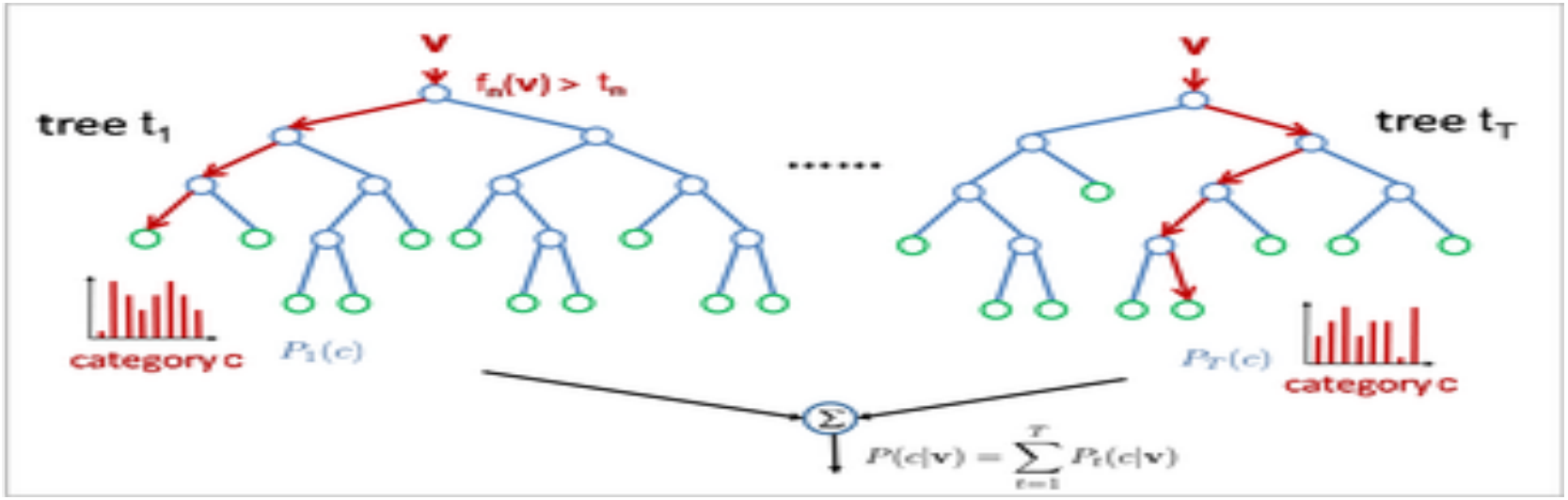
Jie Hu, Department of CSE

Outline

- Random Forests
- Parallelized Implementation
- Experimental Results

Random Forest

- **Random forests** (RF) are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.
- Given a test sample as the input, the output of the random forest is the combination of results from all individual trees



Sequential Random Forest Learning

- Selection of training samples for each tree
- Tree Construction

Algorithm 1 RandomForest(examples $\langle e_1, \ldots, e_n \rangle$, features F, no. of trees t)

1: **for** $i = 1 \ldots t$ **do**
2: **for** $j = 1 \ldots n$ **do**
3: $e'_j \leftarrow e_{rand(1,n)}$
4: $T_i \leftarrow RandomTree(\langle e'_1, \ldots, e'_n \rangle, F$
5: **return** $T_1, \ldots, T_t$

Algorithm 2 RadomTree(examples $<e_1, \ldots e_n>$, features F

1. **if** TerminalNode($<e_1, \ldots e_n>$) **then**
2. T <- MakeLeaf($<e_1, \ldots e_n>$)
3. **else**
4. K <- RandomSplitFunctionPool($<e_1, \ldots e_n>$, F)
5. k <- BestSplit($<e_1, \ldots e_n>$, K)
6. {*El*,*Er*} <- Distribute($<e_1, \ldots e_n>$, k)
7. *Tl* <- RandomTree(*El*, *F*)
8. *Tr* <- RandomTree(*Er*,*F*)
9. *T* <- {*k*, *Tl*, *Tr*}
10. **return** *T*

Parallelized Implementation

- Why parallelize?

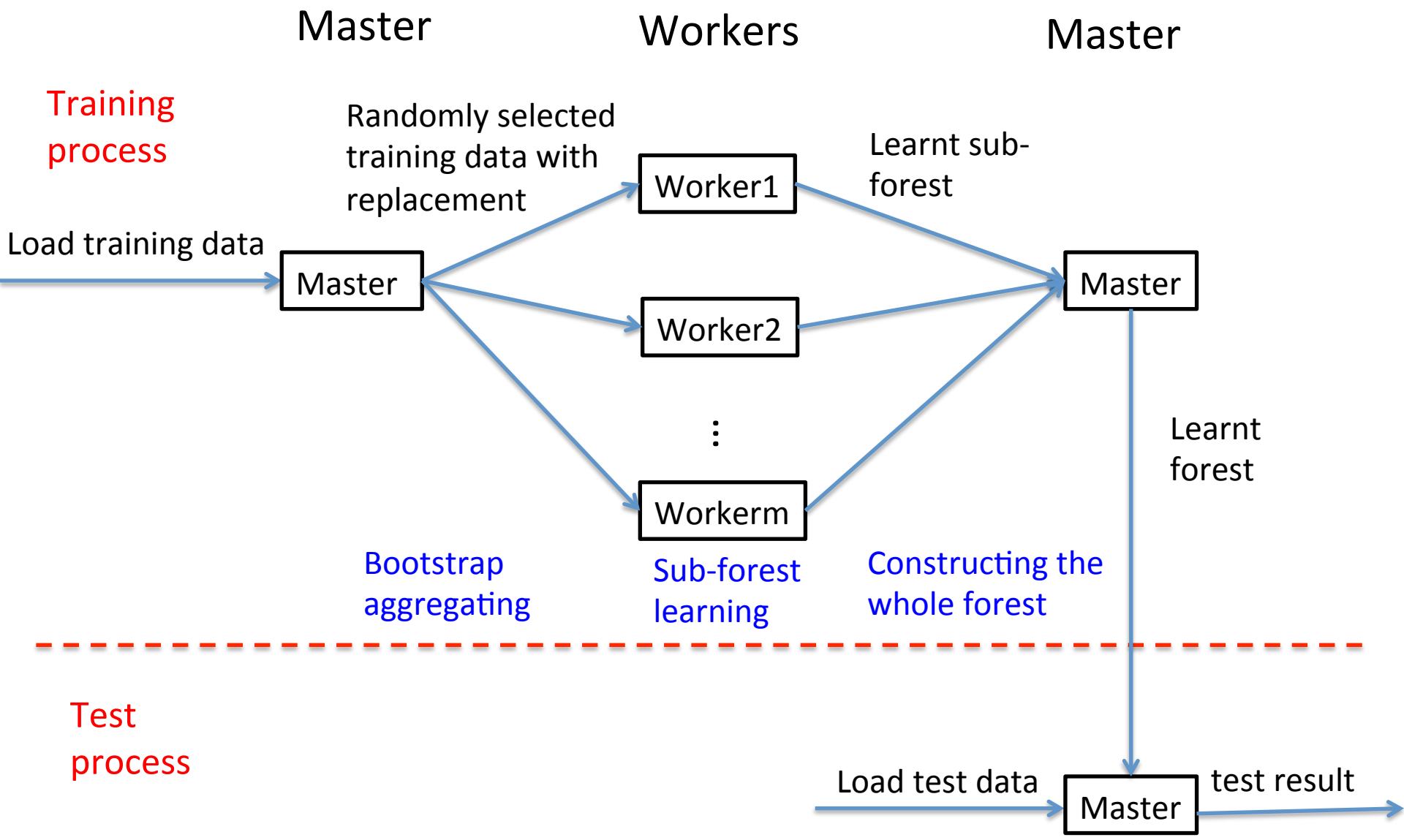
- If we train the random forest sequentially, the time complexity is $O(t|K|n\log n)$.

  t: number of trees K: number of chosen features n: size of training set

- Increasing the number of trees can improve the accuracy of the classifier, but elongate the training time as well.

- Training subsets of trees in the separate processor can largely decrease the training time.

Parallelized Random Forest Learning



Parallelized Random Forest Learning

Master

Algorithm 1 RadomForest(examples $<e_1, \ldots e_n>$, features F), no. of trees t, no. of cores m

1. **for** k=1 … m **do**
2. **for** i=1 … t/m **do**
3. **for** j=1 … n **do**
4. e'_j <- $e_{rand(1,n)}$
5. $T_{(k-1)*t/m+i}$ <= *RandomTree*($<e_1, \ldots e_n>$, F)
6. **return** T_1, …, T_t

Worker

Algorithm 2 RadomTree(examples $<e_1, \ldots e_n>$, features F)

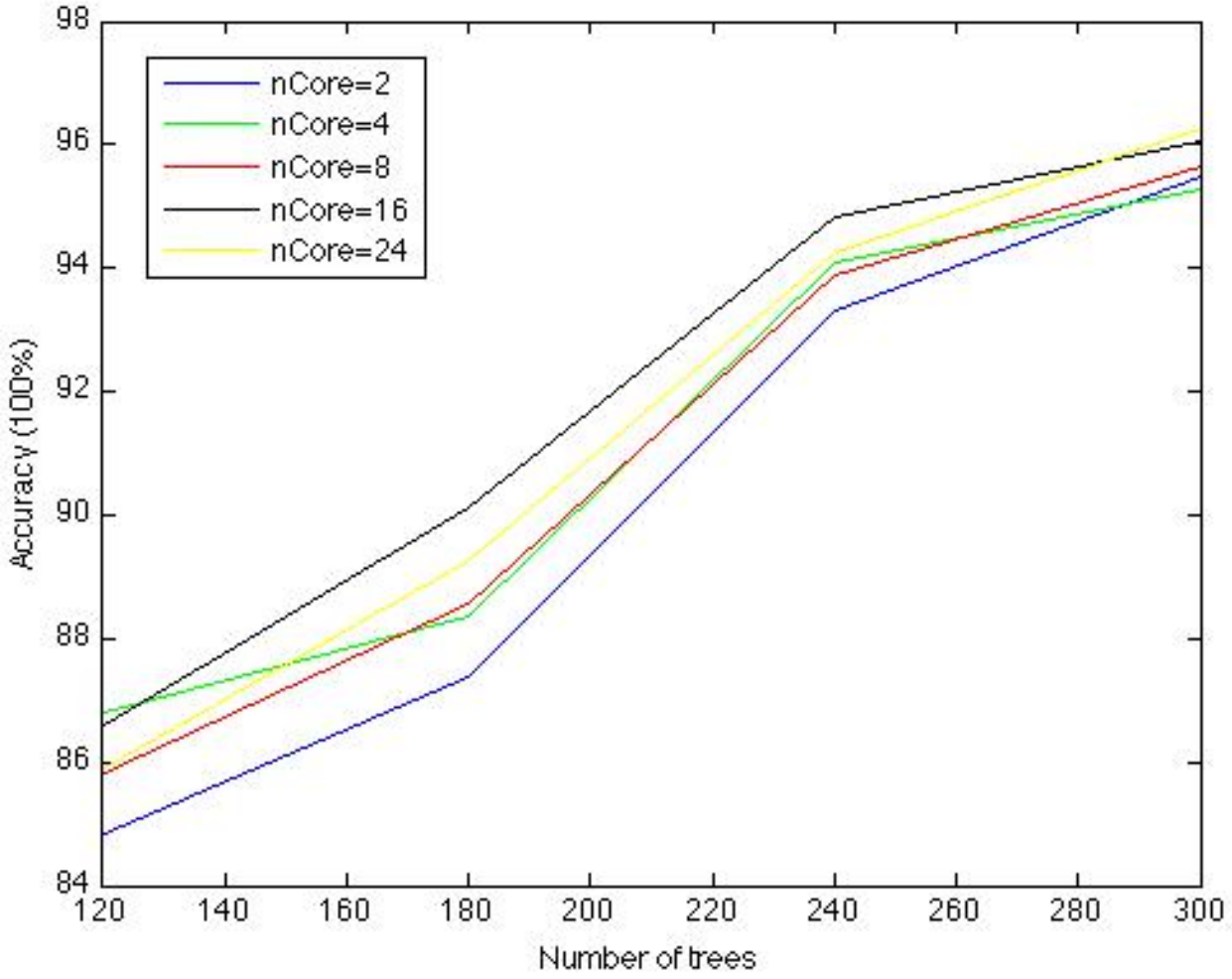
1. **if** TerminalNode($<e_1, \ldots e_n>$) **then**
2. T <- MakeLeaf($<e_1, \ldots e_n>$)
3. **else**
4. *K* <- RandomSplitFunctionPool($<e_1, \ldots e_n>$, F)
5. *k* <- BestSplit($<e_1, \ldots e_n>$, K)
6. {*El*,*Er*} <- Distribute($<e_1, \ldots e_n>$, k)
7. *Tl* <- RandomTree(*El*, *F*)
8. *Tr* <- RandomTree(*Er*,*F*)
9. *T* <- {*k*, *Tl*, *Tr*}
10. **return** *T*

Experiments

- I implement the random forest algorithm using MPI in C
- The random forest is learnt on gene expression data which contains 3 categories and 1363 samples.

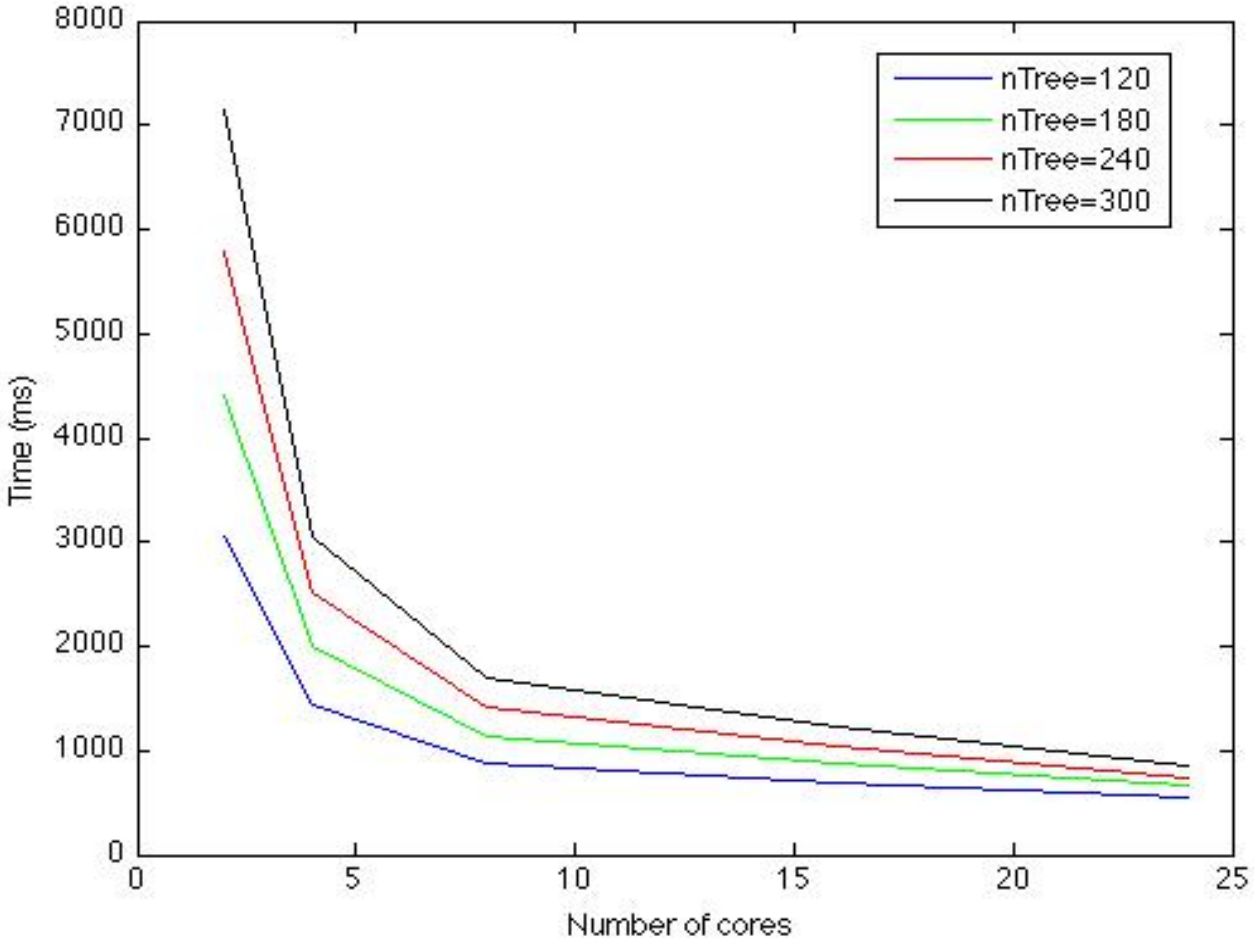
Experimental Results

- Accuracy



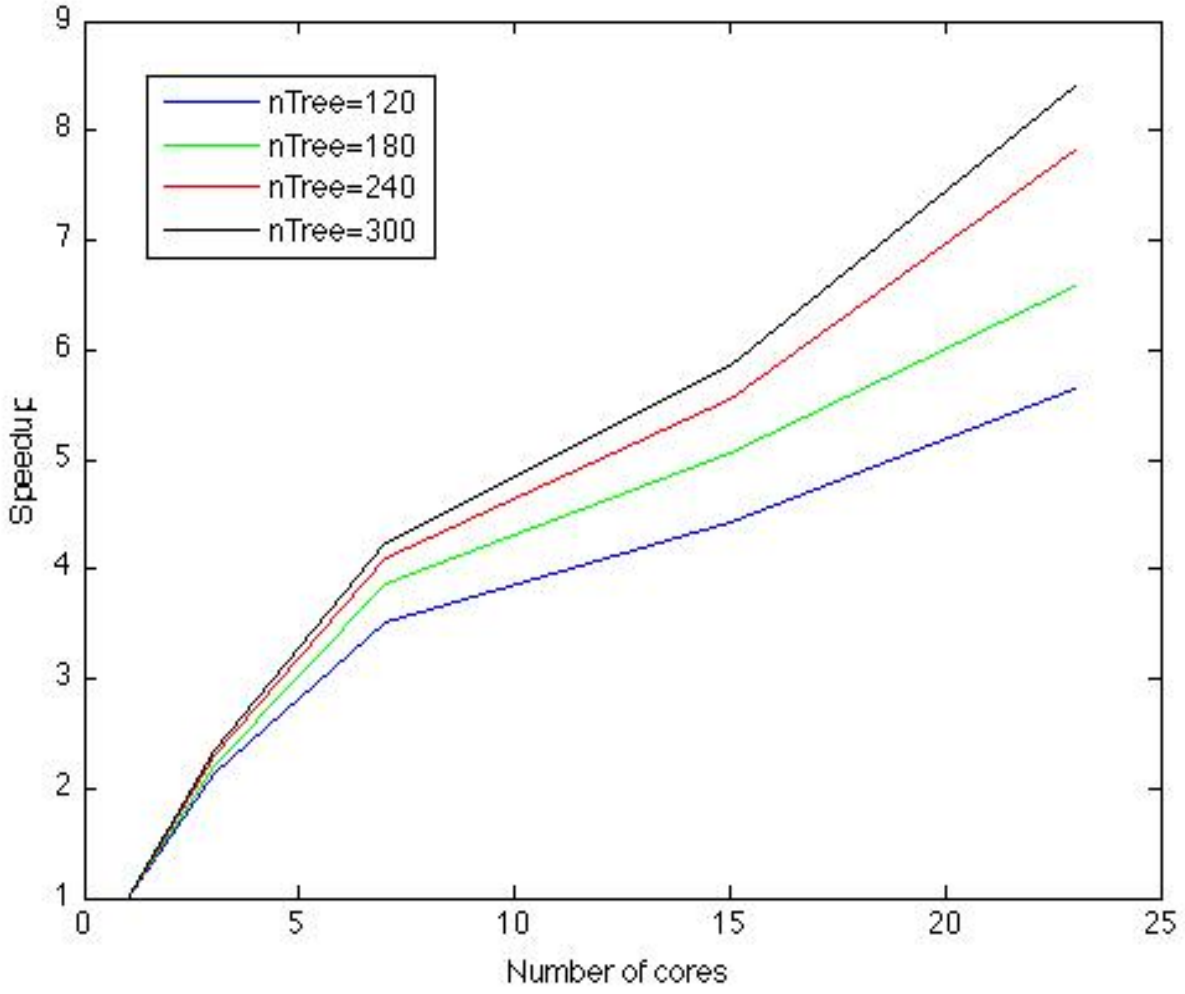
Experimental Results (Contd.)

- Time



Experimental Results(Contd.)

- Speed up



Experimental Results(Contd.)

- Running time (ms)

n_trees n_nodexn_processors	120	180	240	300
1x2	3050	4410	5780	7150
2x2	1430	2010	2520	3060
4x2	870	1140	1410	1690
8x2	690	870	1040	1220
2x8	390	500	600	720
12x2	540	670	740	850
2x12	220	290	360	390

Conclusion and Future work

- Random forest learning is implemented using C in MPI.
- By using parallel methods, we can improves the accuracy of the classification using less time.
- We can apply this parallel methods on larger dataset and try to parallelize the construction for each decision tree.

Thank you. Q & A