Weekly Report
03/29/2015

Jingyu Deng

This Week

- Wrote data generator
- Implemented a single estimator algorithm for comparison
- Generated data for experiments
- Tested those algorithms on those data

Data Set

- I followed the rule introduced in Wang's paper to generate experiment data.
- There are 3 parameters:
    - ***d***: the total number of dimensions
    - ***k***: the number of changing dimensions
    - ***t***: the magnitude of change
- To simulate concept drift when generating data, the weights on ***k*** changing dimensions continuously change ***t*** every 1000 data points.
- In our experiments:
    - ***d*** is 10 (same as ***d*** in Wang's paper)
    - ***k*** is in [2, 4, 6, 8] and we can simply regard ***k*** as the difficulty of predicting when concept drift occurs.
    - **t** is in [0.1, 0.2, 0.3, … , 1.0]. I tested on some data file and found that this parameter didn't play an important role. So in the experiment I only tested data with t=0.1
- In naming of data file, d10_k0.2_t0.1 means ***d***=10, ***k***=10*0.2=2, ***t***=0.1

Tested Algorithms

- In the experiments, 3 algorithms are tested:
    - 1) Anil's algorithm: it keeps ***n*** decision trees and maintains weights on them. Sometimes it discards several decision trees and replaces them by new trees based on the latest data points whose size is ***chunk_size***.
    - 2) Wang's algorithm: it keeps also ***n*** decision trees. However it trained a new tree every ***chunk_size*** data points. And it always keeps top ***n*** decision trees based to their weights. Weights are calculated only when a new data chunk is coming.
    - 3) Single algorithm: it always trains a new decision tree based on the last ***n * chunk_size*** data points every ***chunk_size*** points. And it will keep only this new tree for next prediction.
- Thus in the experiment, I tested all 3 algorithms performances under different ***n*** and ***chunk_size***.
- ***n*** is in [1, 2, 4, 6, 8, 10, 12, 20]
- ***chunk_size*** is in [250, 500, 750, 1000]

Experiment Parameters

Parameter	Description	Range
d	the total number of dimensions	10
k	the number of changing dimensions	[0.2, 0.4, 0.6, 0.8]
t	the magnitude of the change	0.1
n	the number of trees(our, wang) the number of chunks used for training(single)	[1, 2, 4, 6, 8, 10, 12, 20]
chunk_size	the size of chunk	[250, 500, 750, 1000]

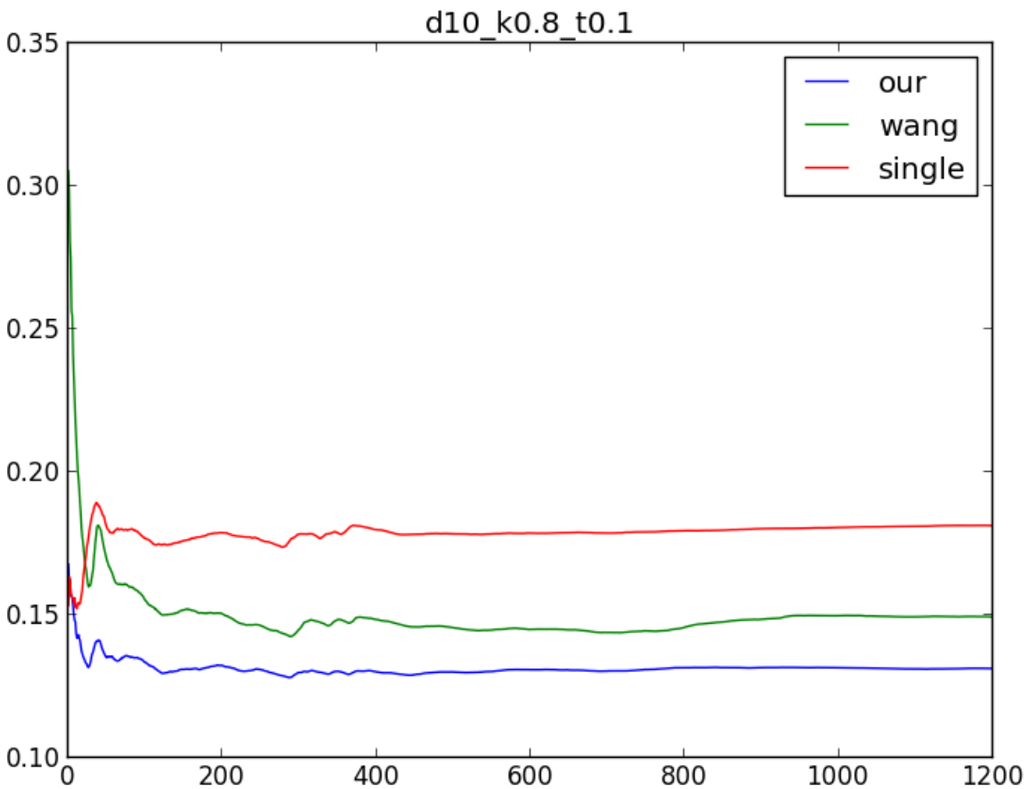
Experiment Results

- I uploaded all results onto Google Drive.
- Final error rates of each algorithm are stored in txt file.
    - E.g. d10_k0.2_t0.1.txt stores all final error rates for the data set d10_k0.2_t0.1
    - Each line is in the format as:
        - 250(chunk_size) 1(n=1) 0.432791111111(error rate) O(our algorithm)
- I also imported them into .numbers file (for mac) and .xlsx file (for windows). You can also access them on Google Drive.

	chuk_size	n	error_rate	algo
2	250	1	0.432791111111	O
3	500	1	0.429337777778	O
4	750	1	0.382234444444	O
5	1000	1	0.362382222222	O
6	250	2	0.130616666667	O
7	500	2	0.401296666667	O
8	750	2	0.271946666667	O
9	1000	2	0.270197777778	O
10	250	4	0.133837777778	O
11	500	4	0.114472222222	O
12	750	4	0.07817	O
13	1000	4	0.125988888889	O
14	250	6	0.0893366666667	O
15	500	6	0.0677433333333	O
16	750	6	0.0652966666667	O
17	1000	6	0.07591	O
18	250	8	0.0645766666667	O
19	500	8	0.0600077777778	O
20	750	8	0.0557477777778	O
21	1000	8	0.0616833333333	O

Experiment Results

- All plots are stored in the folder whose name indicates the data set.
    - E.g. in folder d10_k0.2_t0.1, the picture ch1000_n1.png shows the plot of error rate for 3 algorithms under d=10, k=0.2, t=0.1, chunk_size=1000, n=1
    - X axis represents the number of data points processed. Y axis represents the error rate.



d10_k0.8_t0.1/ch750_n12.png

Experimental Phenomena

- I summarized some interesting experimental phenomenas shown in the next few slides.

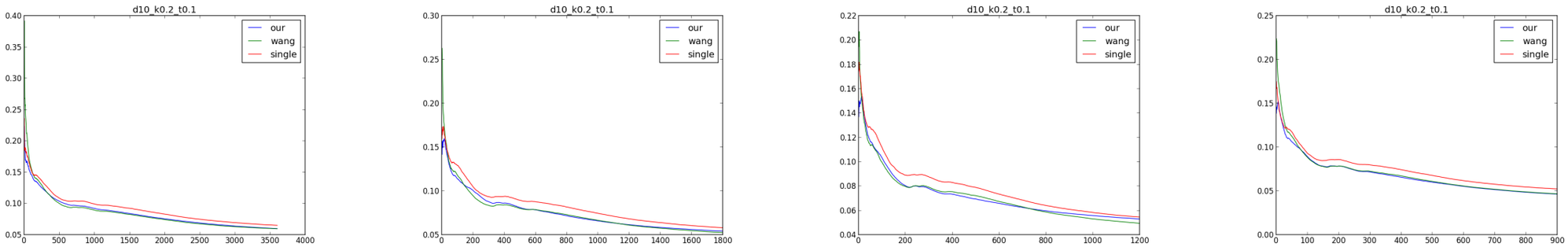
Phenomena 1

- As ***k*** grows (more difficult to predict), the gap between single algorithm and the other 2 (our, wang) becomes larger and larger. Red line represents the single algorithm.

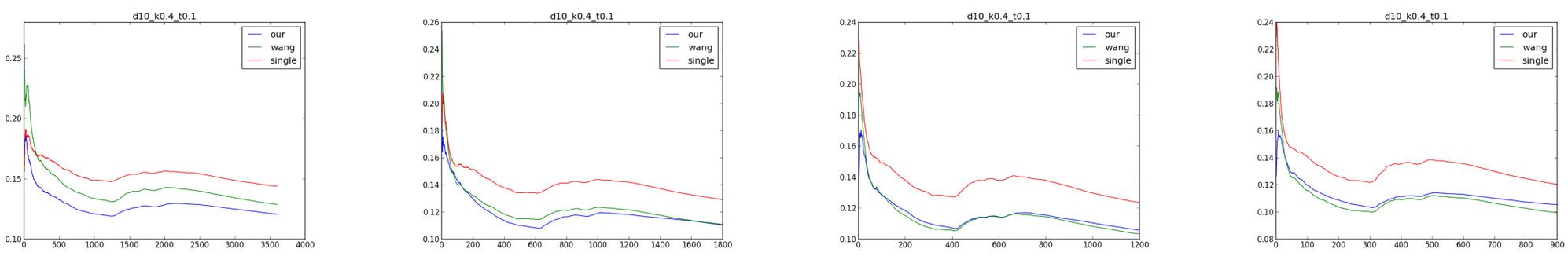
n=10 for all

ch=250 ch=500 ch=750 ch=1000

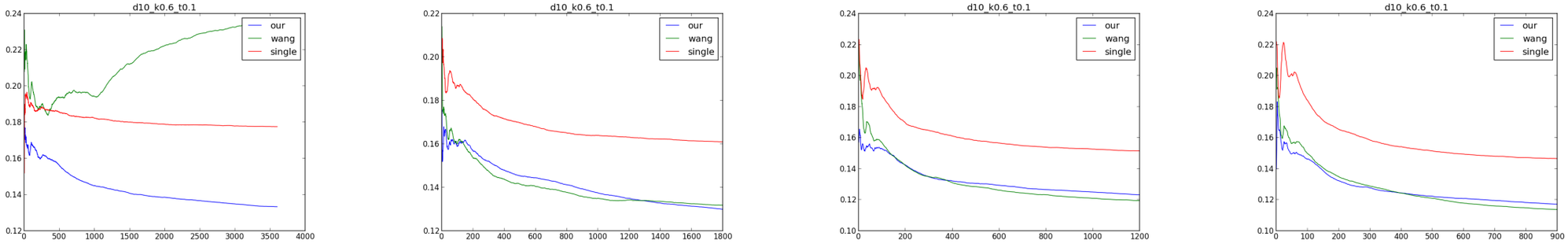
k=2



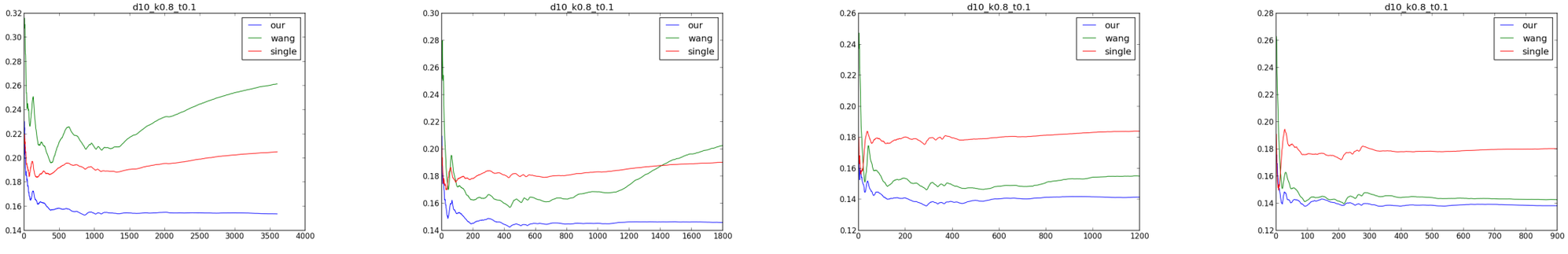
k=4



k=6



k=8



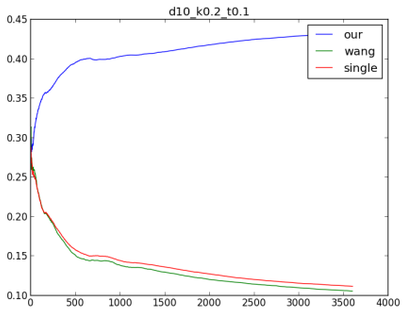
Phenomena 2

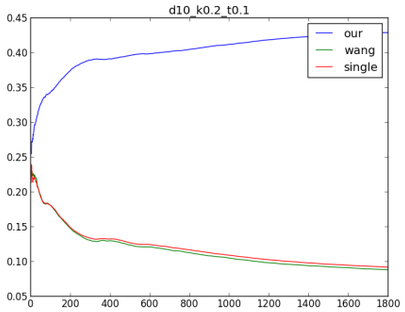
- Our algorithm works really badly when ***n*** is small. The reason is that under our replace policy, which uses a single threshold, no trees have the possibility to be replaced when ***n*** is small. But it's easy to improve. :)

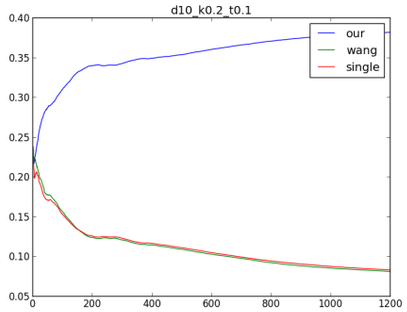
n=1

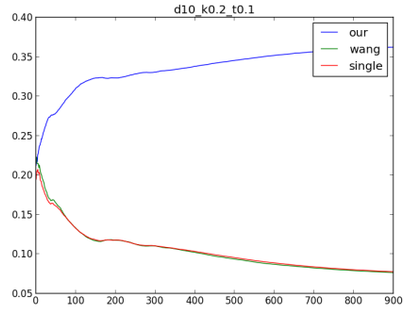
ch=250 ch=500 ch=750 ch=1000

k=2

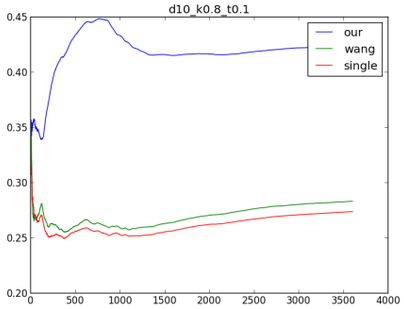


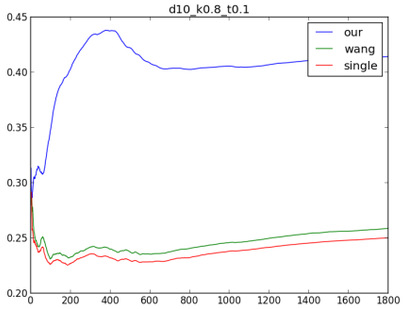


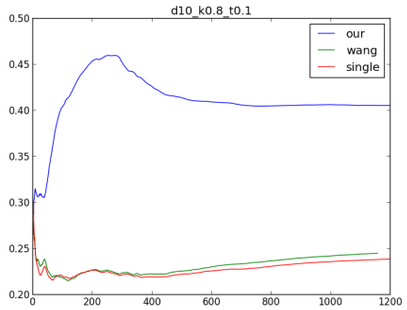


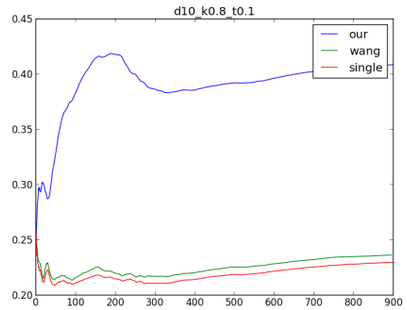


k=8



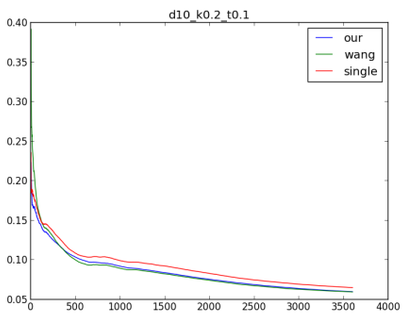


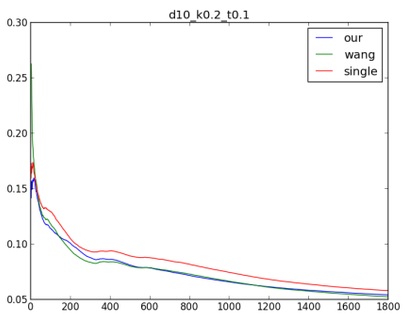


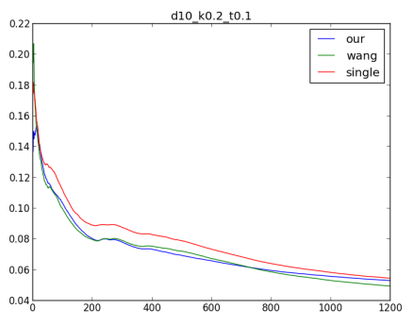


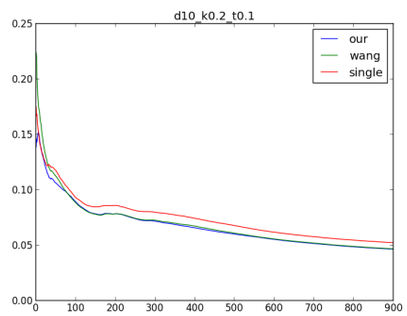
n=10

k=2

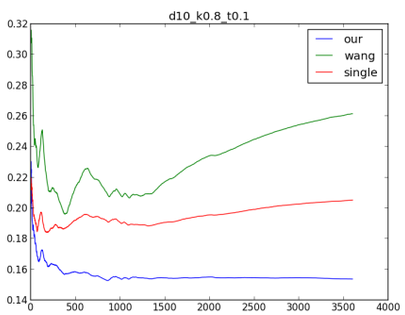


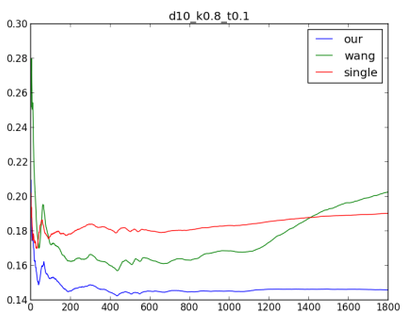


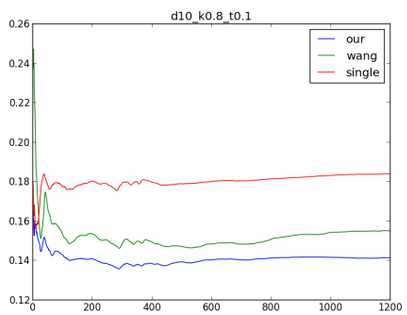


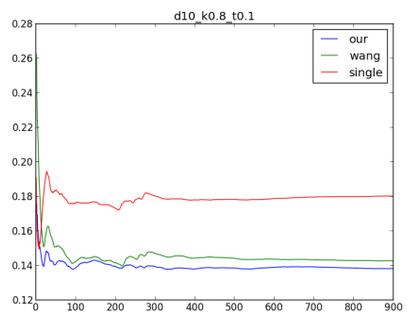


k=8



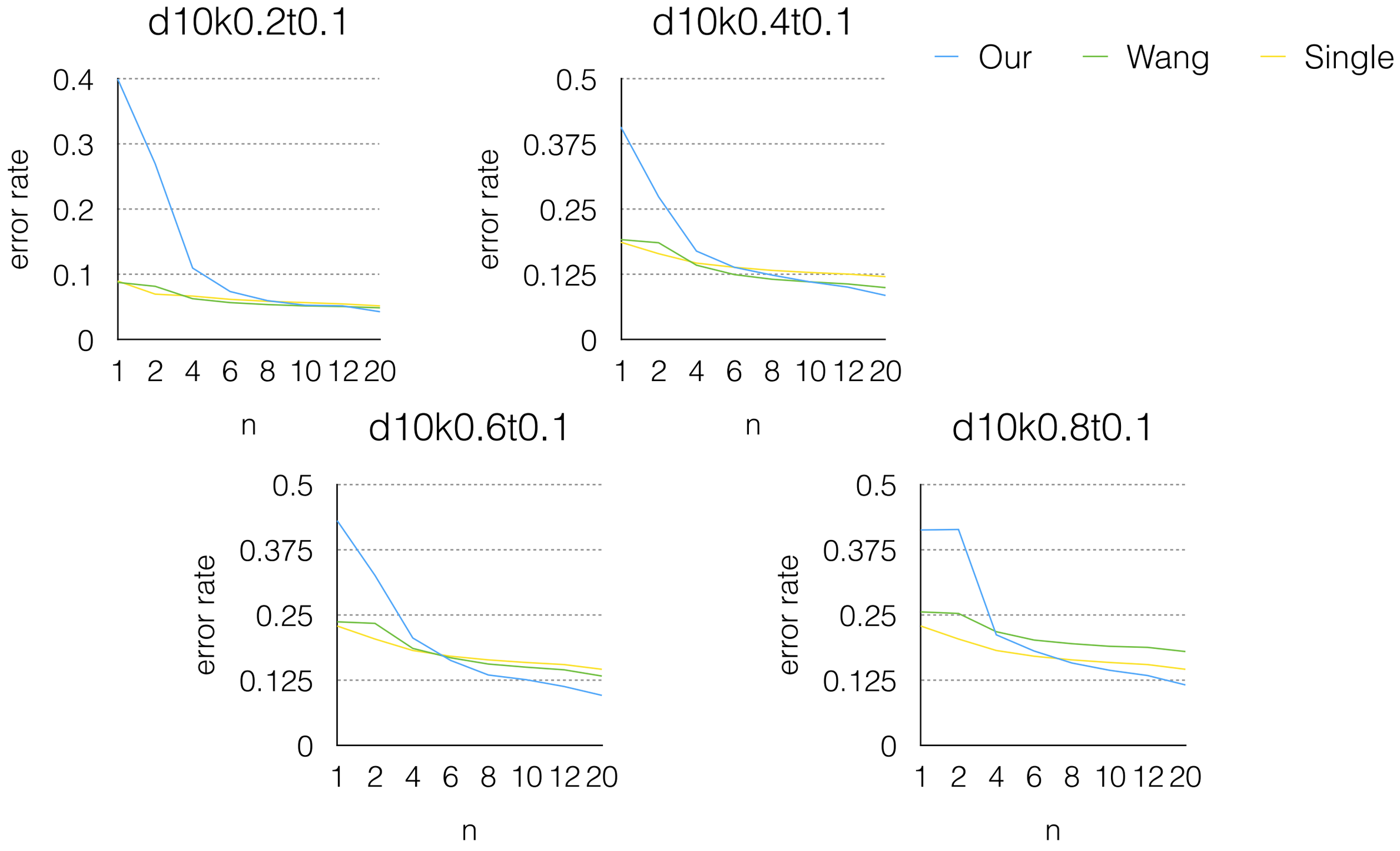




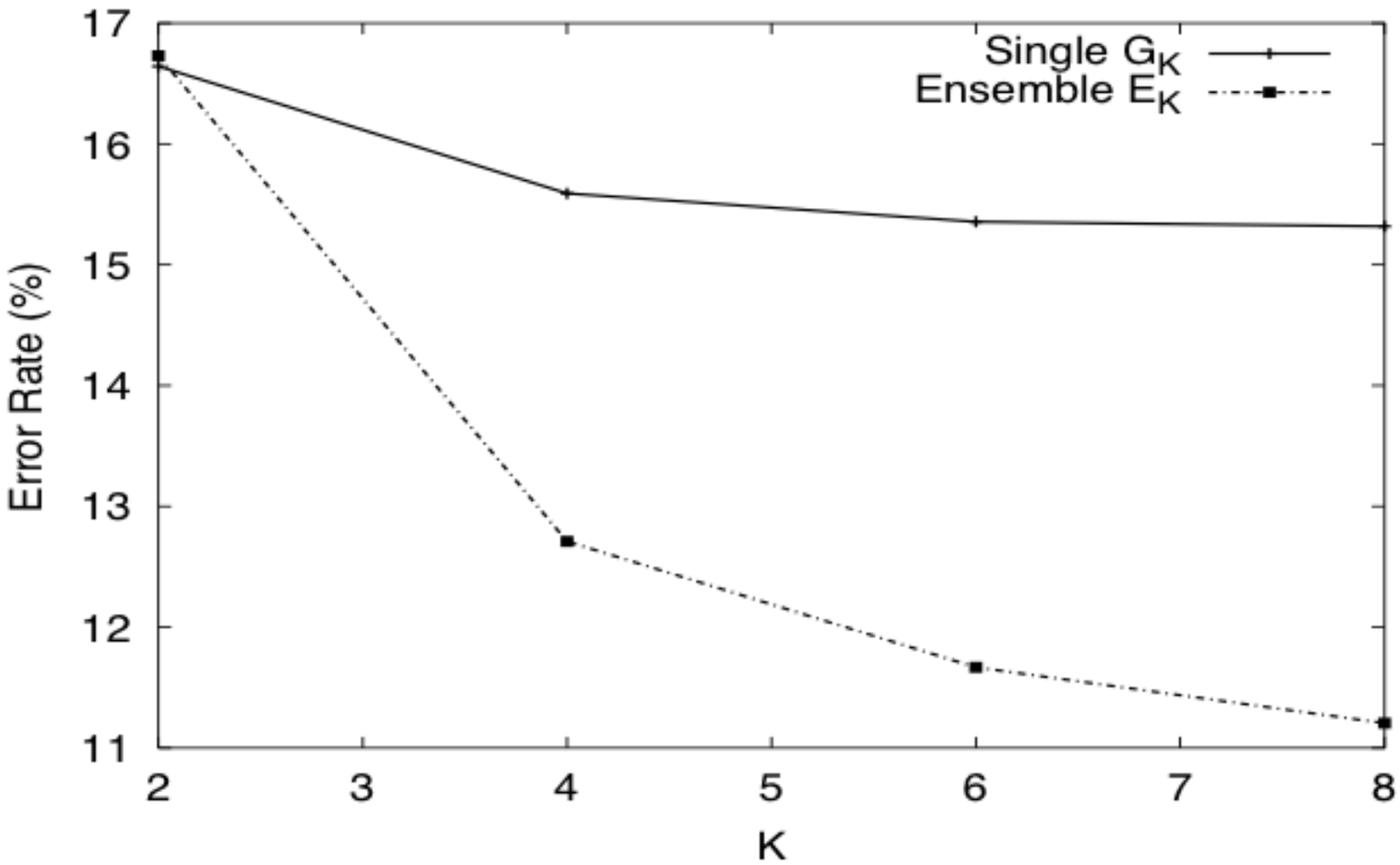


Phenomena 3

- As ***n*** increases, the error rates of all 3 algorithms (especially ours) drop.



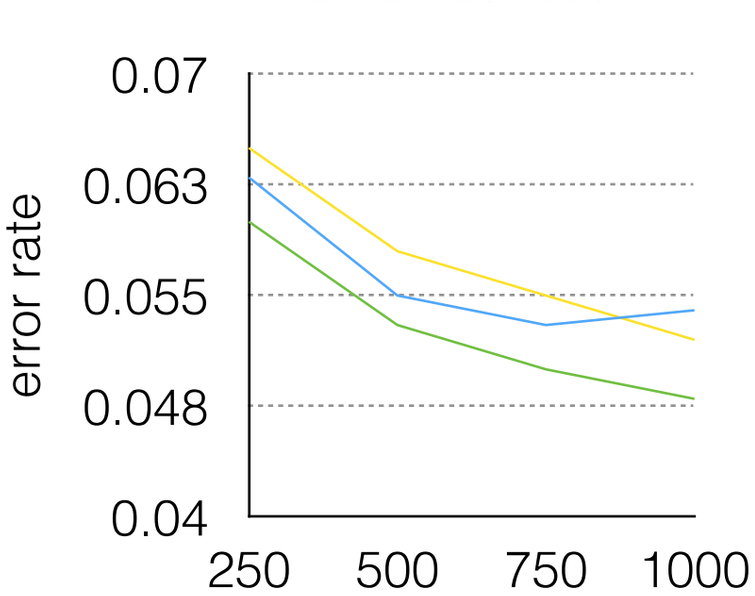
- In Wang's paper, they also mention this phenomena which the error rate drops while the number of classifiers increases.

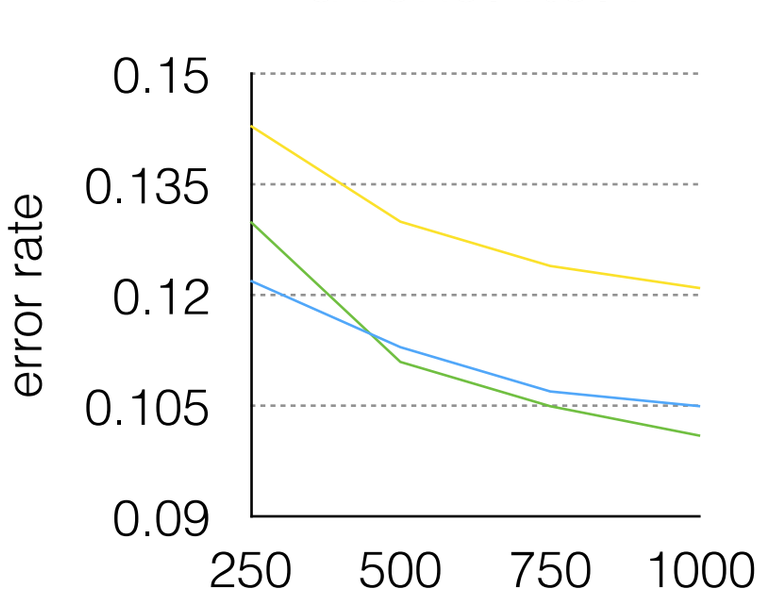


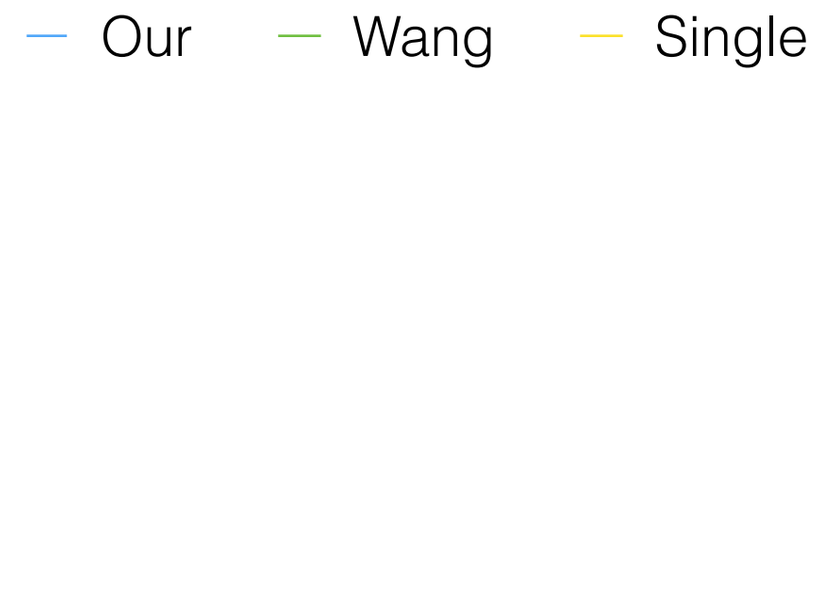
(a) Varying window size/ensemble size

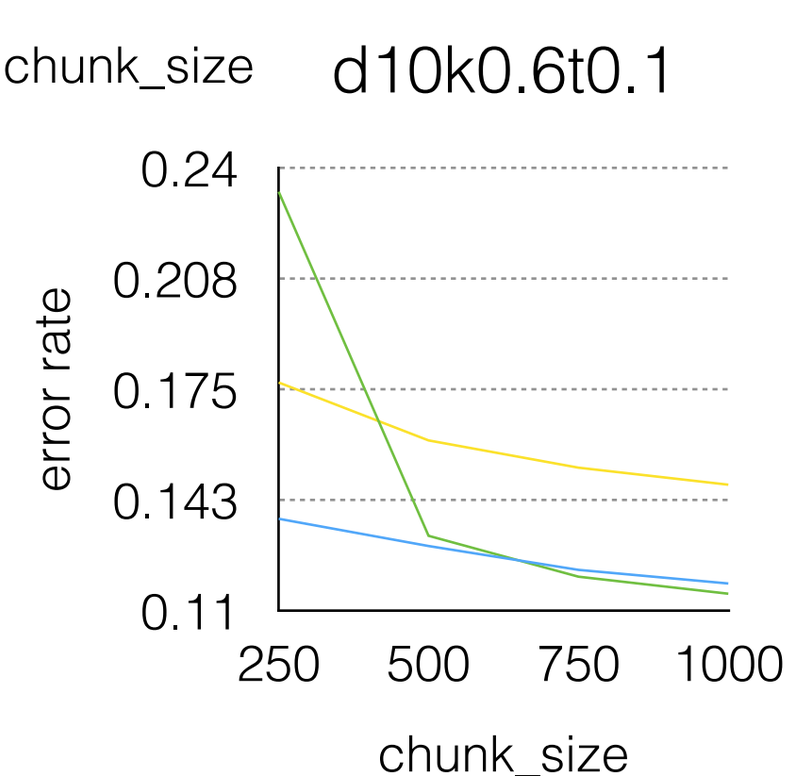
Phenomena 4

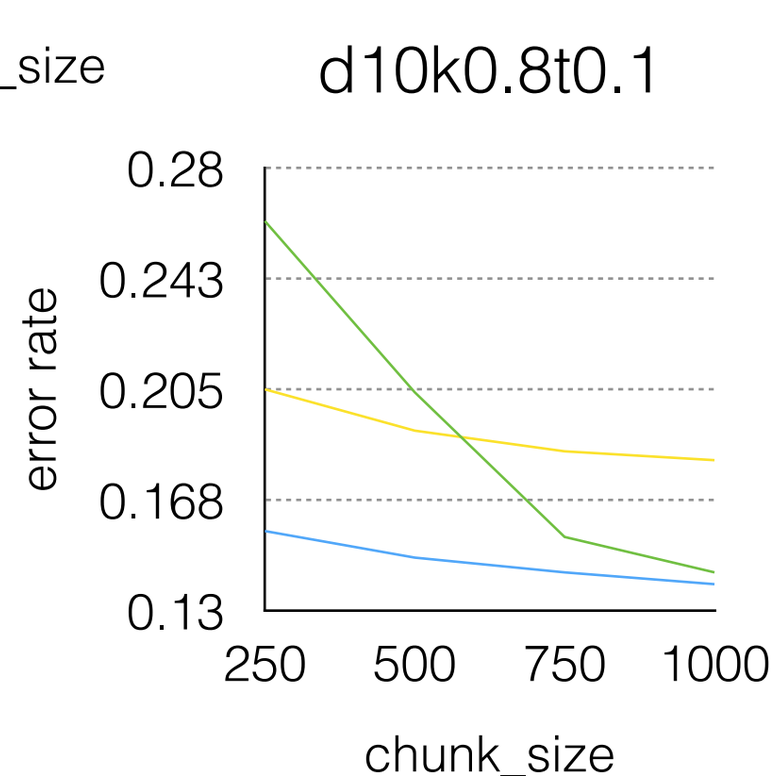
- As the ***chunk_size*** increases, performances of all 3 algorithms, (especially Wang's) are getting better.







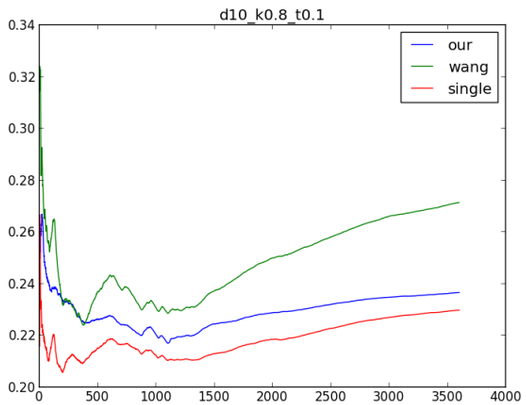


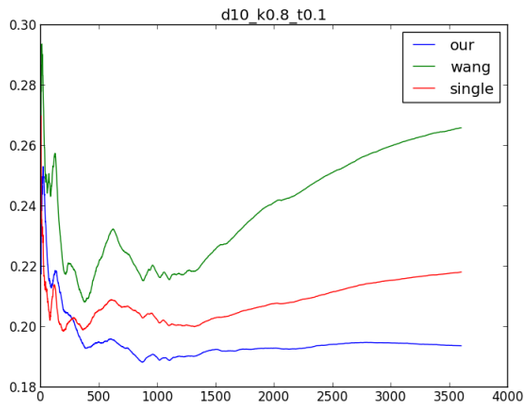


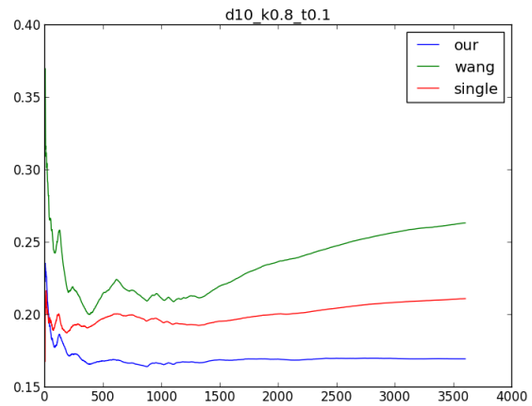
- Wang's algorithm work terribly when ***chunk_size*** is small. Here are some plots with ***k***=0.8, ***chunk_size***=250,500

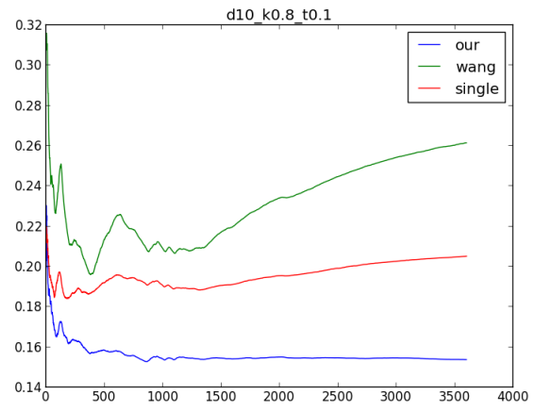
n=4 n=6 n=8 n=10

chunk_size =250

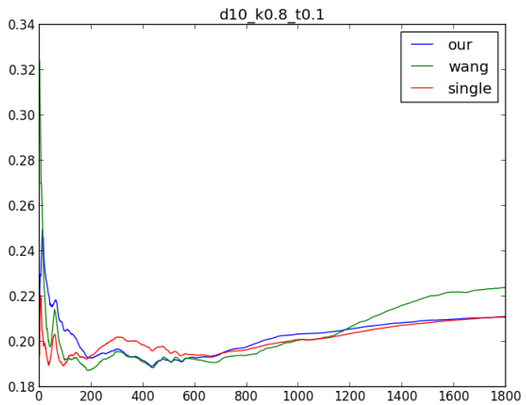


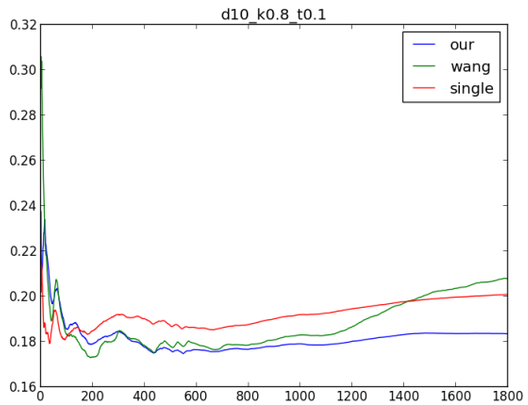


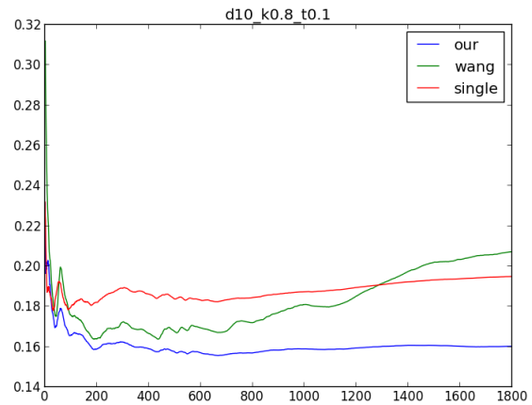


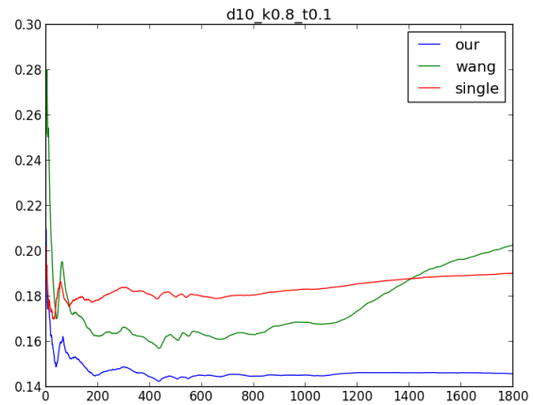


chunk_size =500

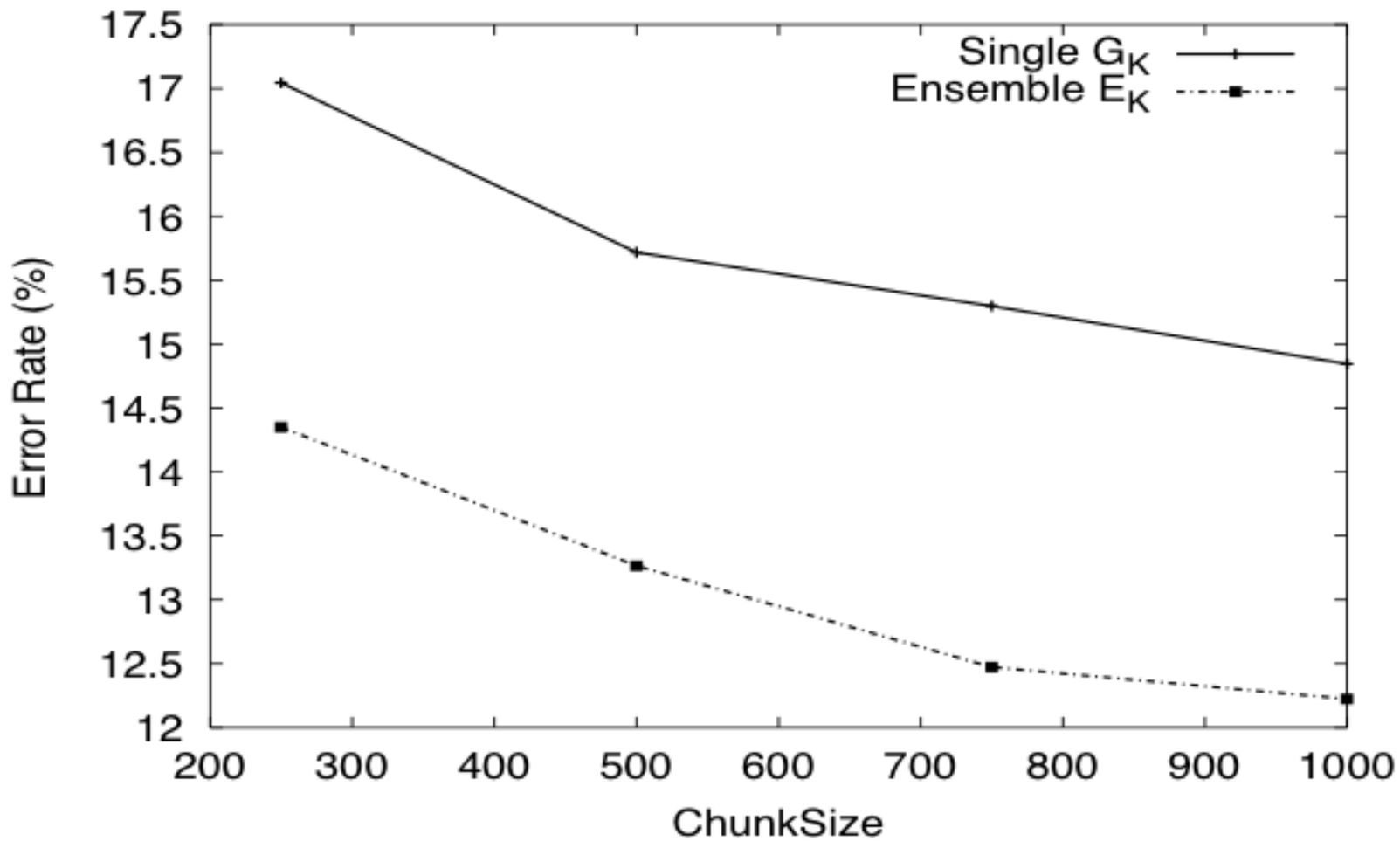








- In Wang's paper, they also notice this relation between chunk_size and error_rate.



(b) Varying `ChunkSize`

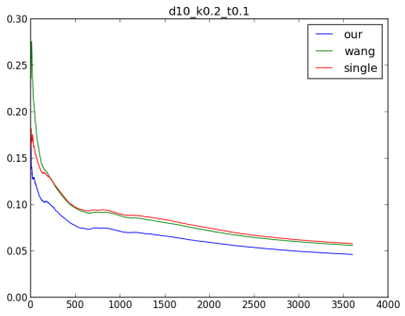
Phenomena 5

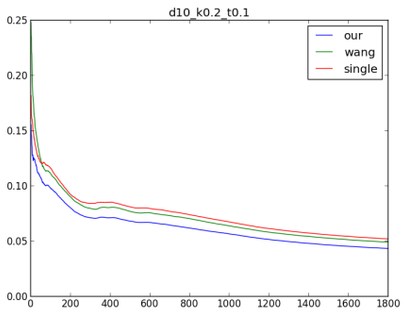
- If n is big enough, our algorithm always gets the **best** performance for all tested data. Here are some plot evidence with n=20. The tests with n=32 are still running and I believe our algorithm will get much better results than the other two.

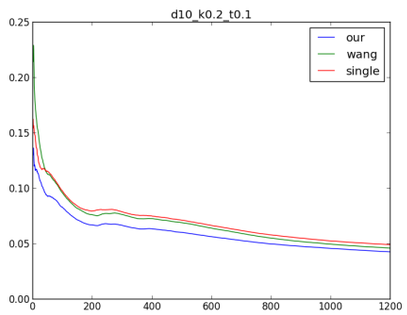
n=20 for all

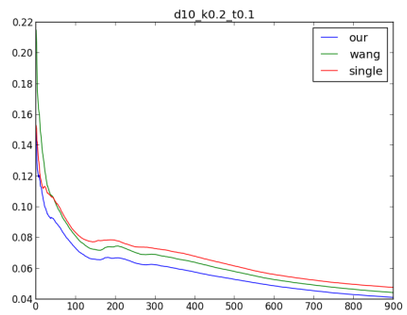
ch=250 ch=500 ch=750 ch=1000

k=2

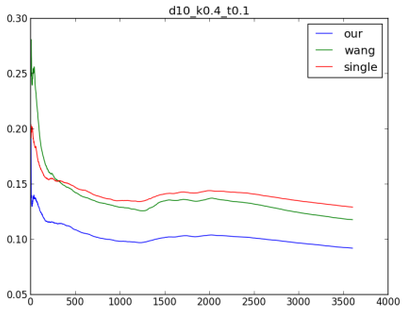


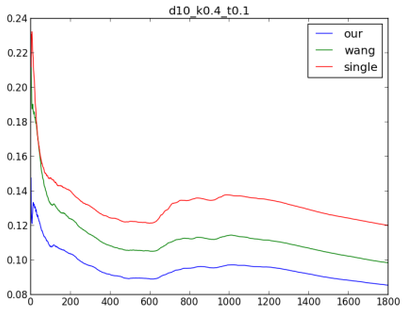


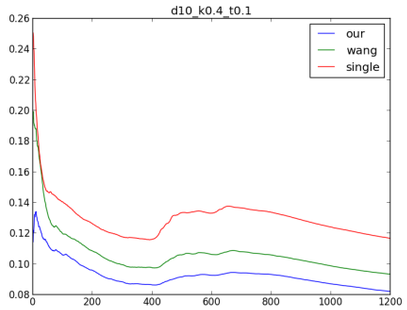


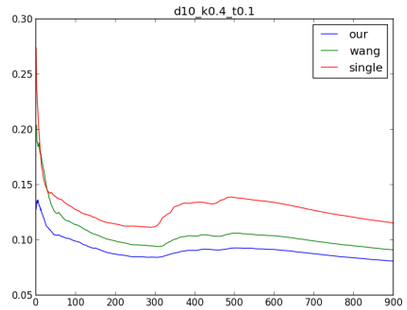


k=4

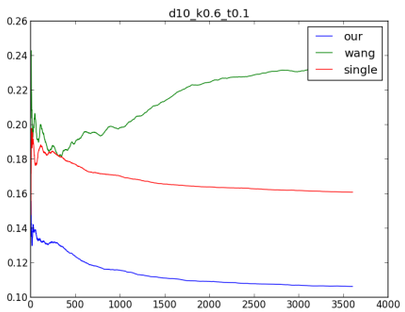


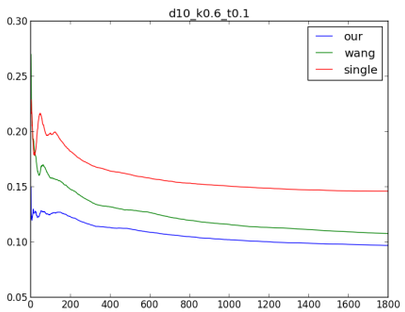


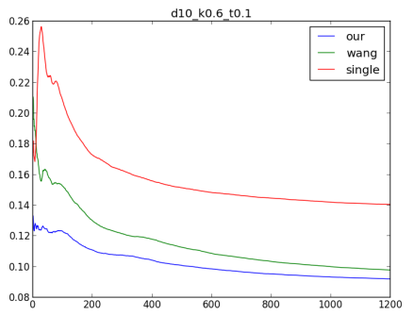


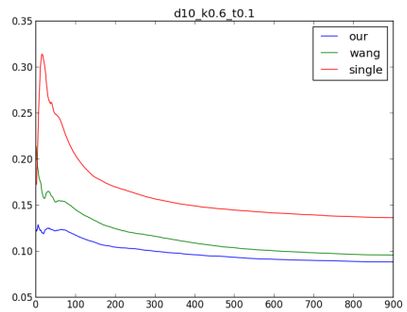


k=6

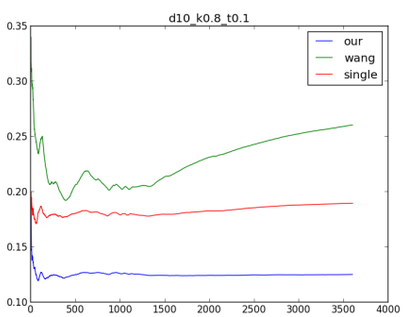


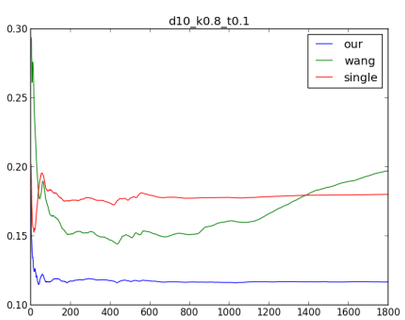


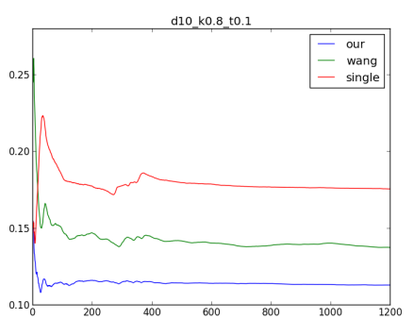


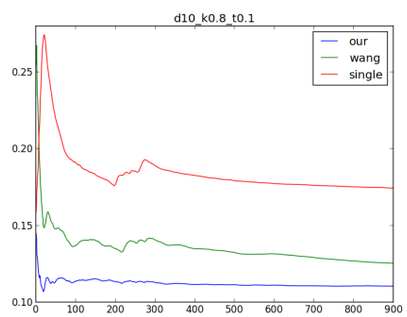


k=8









Time Analysis

- I forgot to record the running time during all tests. But I tested a few data. The single algorithm is faster than the other two. And ours has the same running time as Wang's. Actually the speed of our algorithm depends on the replace threshold. The lower the threshold is, the more trees will be replaced during and the more time will be cost.

Summary of Parameter Influence

	k	chunk_size	n
Ours	As k increases, the error rate increases too.		Terrible performance when n is small. But as n becomes big enough, it always gets the best results.
Wang's		Terrible performance when chunk_size is small	As n increases, the error rate drops.
Single	Work badly as k becomes bigger.		

Next Step

- Modify the replace policy to adapt to the situation when n is small.
- Do more experiments.
- Try other complicated models?

Thank you