119010344
Xiao Nan

Cloud Computing hw3: Problem 1-4:

1. (a) $P(Yes) = \frac{2}{5}$, $P(No) = \frac{3}{5}$

(b)

$P(Bad \mid Yes) = \frac{1}{2}$	$P(Single \mid Yes) = \frac{1}{2}$	$P(High \mid Yes) = \frac{1}{2}$
$P(Good \mid Yes) = \frac{1}{2}$	$P(Married \mid Yes) = \frac{1}{2}$	$P(Low \mid Yes) = \frac{1}{2}$
$P(Bad \mid No) = \frac{2}{3}$	$P(Single \mid No) = \frac{2}{3}$	$P(High \mid No) = \frac{1}{3}$
$P(Good \mid No) = \frac{1}{3}$	$P(Married \mid No) = \frac{1}{3}$	$P(Low \mid No) = \frac{2}{3}$

(c) $P(A_1, A_2, A_3 \mid Yes) = P(A_1 \mid Yes) \cdot P(A_2 \mid Yes) \cdot P(A_3 \mid Yes)$

$P(A_1, A_2, A_3 \mid No) = P(A_1 \mid No) \cdot P(A_2 \mid No) \cdot P(A_3 \mid No)$

(d) $P(Yes \mid A_1, A_2, A_3) = \frac{P(A_1, A_2, A_3 \mid Yes) \cdot P(Yes)}{P(A_1, A_2, A_3)}$

$= \frac{P(A_1 \mid Yes) \cdot P(A_2 \mid Yes) \cdot P(A_3 \mid Yes) \cdot P(Yes)}{P(A_1, A_2, A_3)}$

$P(No \mid A_1, A_2, A_3) = \frac{P(A_1, A_2, A_3 \mid No) \cdot P(No)}{P(A_1, A_2, A_3)}$

$= \frac{P(A_1 \mid No) \cdot P(A_2 \mid No) \cdot P(A_3 \mid No) \cdot P(No)}{P(A_1, A_2, A_3)}$

(cont. at the next page)

(e) A_1=Bad, A_2=Married, A_3=Low:

$$P(A_1=\text{Bad}, A_2=\text{Married}, A_3=\text{low} \mid \text{No}) \cdot P(\text{No}) = P(A_1=\text{Bad} \mid \text{No}) \cdot P(A_2=\text{Married} \mid \text{No}) \cdot P(A_3=\text{low} \mid \text{No}) \cdot P(\text{No})$$

$$= \frac{2}{3} \cdot \frac{1}{3} \cdot \frac{2}{3} \cdot \frac{3}{5}$$

$$= \frac{4}{45}$$

$$P(A_1=\text{Bad}, A_2=\text{Married}, A_3=\text{low} \mid \text{Yes}) \cdot P(\text{Yes}) = P(A_1=\text{Bad} \mid \text{Yes}) \cdot P(A_2=\text{Married} \mid \text{Yes}) \cdot P(A_3=\text{low} \mid \text{Yes}) \cdot P(\text{Yes})$$

$$= \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{2}{5}$$

$$= \frac{1}{20}$$

$$\therefore \frac{4}{45} > \frac{1}{20}$$

$$P(A_1=\text{Bad}, A_2=\text{Married}, A_3=\text{low} \mid \text{No}) \cdot P(\text{No}) > P(A_1=\text{Bad}, A_2=\text{Married}, A_3=\text{low} \mid \text{Yes}) \cdot P(\text{Yes})$$

$$\frac{P(A_1=\text{Bad}, A_2=\text{Married}, A_3=\text{low} \mid \text{No}) \cdot P(\text{No})}{P(A_1=\text{Bad}, A_2=\text{Married}, A_3=\text{low})} > \frac{P(A_1=\text{Bad}, A_2=\text{Married}, A_3=\text{low} \mid \text{Yes}) \cdot P(\text{Yes})}{P(A_1=\text{Bad}, A_2=\text{Married}, A_3=\text{low})}$$

$$P(\text{No} \mid A_1=\text{Bad}, A_2=\text{Married}, A_3=\text{low}) > P(\text{Yes} \mid A_1=\text{Bad}, A_2=\text{Married}, A_3=\text{low})$$

According to the past data, the probability for refusing loan for the person is higher

$\therefore$ The loan should be denied

2.(a) $d_{a\to d} = ||(1.1, 3.1) - (1.4, 3.6)||^2 = 0.34$

$d_{a\to e} = ||(1.1, 3.1) - (2.4, 5.6)||^2 = 7.94$

$d_{a\to f} = ||(1.1, 3.1) - (3.1, 3.4)||^2 = 4.09$

$d_{b\to d} = ||(1.9, 5.5) - (1.4, 3.6)||^2 = 3.86$

$d_{b\to e} = ||(1.9, 5.5) - (2.4, 5.6)||^2 = 0.26$

$d_{b\to f} = ||(1.9, 5.5) - (3.1, 3.4)||^2 = 5.85$

$d_{c\to d} = ||(2.8, 6.4) - (1.4, 3.6)||^2 = 7.84$

$d_{c\to e} = ||(2.8, 6.4) - (2.4, 5.6)||^2 = 0.8$

$d_{c\to f} = ||(2.8, 6.4) - (3.1, 3.4)||^2 = 9.09$

$d_{g\to d} = ||(2.7, 3.3) - (1.4, 3.6)||^2 = 1.78$

$d_{g\to e} = ||(2.7, 3.3) - (2.4, 5.6)||^2 = 5.38$

$d_{g\to f} = ||(2.7, 3.3) - (3.1, 3.4)||^2 = 0.17$

∴ d-centered cluster: a

e-centered cluster: b, c

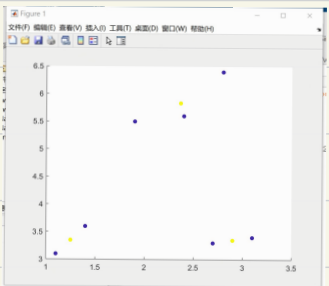
f-centered cluster: g

$\bar{x}_d' = \left(\frac{1.4+1.1}{2}, \frac{3.6+3.1}{2}\right) = (1.25, 3.35)$

$\bar{x}_e' = \left(\frac{1.9+2.8+2.4}{3}, \frac{5.5+6.4+5.6}{3}\right) = \left(\frac{71}{30}, \frac{35}{6}\right)$

$\bar{x}_f' = \left(\frac{3.1+2.7}{2}, \frac{3.4+3.3}{2}\right) = (2.9, 3.35)$

(b)



By visual inspection:

Cluster1 : a, d (with new centroid d')

Cluster2 : e, b, c (with new centroid e')

Cluster3 : f, g (with new centroid f')

$\bar{x}''_{d'} = \left(\frac{1.4+1.1}{2}, \frac{3.6+3.1}{2}\right) = (1.25, 3.35)$

$\bar{x}''_{e'} = \left(\frac{1.9+2.8+2.4}{3}, \frac{5.5+6.4+5.6}{3}\right) = \left(\frac{71}{30}, \frac{35}{6}\right)$

$\bar{x}''_{f'} = \left(\frac{3.1+2.7}{2}, \frac{3.4+3.3}{2}\right) = (2.9, 3.35)$

Next iteration:

$d'_{a\to d'} < \min\{d'_{a\to e'}, d'_{a\to f'}\}$

$d'_{d\to d'} < \min\{d'_{d\to e'}, d'_{d\to f'}\}$

$d'_{e\to e'} < \min\{d'_{e\to d'}, d'_{e\to f'}\}$

$d'_{b\to e'} < \min\{d'_{b\to d'}, d'_{b\to f'}\}$

$d'_{c\to e'} < \min\{d'_{c\to d'}, d'_{c\to f'}\}$

$d'_{f\to f'} < \min\{d'_{f\to e'}, d'_{f\to d'}\}$

$d'_{g\to f'} < \min\{d'_{g\to e'}, d'_{g\to d'}\}$

$\Rightarrow$ Cluster1 : a, d (with new centroid d'')

Cluster2 : e, b, c (with new centroid e'')

Cluster3 : f, g (with new centroid f'')

Which are unchanged

$$\begin{cases} \bar{X}_{\alpha}^{(11)} = \left(\frac{1.4+1.1}{2}, \frac{3.6+3.1}{2}\right) = (1.25, 3.35) \\ \bar{X}_{e}^{(11)} = \left(\frac{1.9+2.8+2.4}{3}, \frac{5.5+6.4+5.6}{3}\right) = \left(\frac{71}{30}, \frac{35}{6}\right) \\ \bar{X}_{\beta}^{(11)} = \left(\frac{3.1+2.7}{2}, \frac{3.4+3.3}{2}\right) = (2.9, 3.35) \end{cases}$$

Which are unchanged

∴ The clustering process converges

with centroids. $C(1.25, 3.35), \left(\frac{71}{30}, \frac{35}{6}\right), (2.9, 3.35)$

3.(a)(1) Denote the Filter size : $m \times n$:

$$28 = \left| \frac{32 - m}{1} \right| + 1$$

(input length; Stride)

$$\Rightarrow m = 5$$

$$28 = \left| \frac{32 - n}{1} \right| + 1$$

(input width; Stride)

$$\Rightarrow n = 5$$

$\therefore$ Filter size $= 5 \times 5$

(2) # neurons $= 6 \times (28 \times 28) = 4704$

(6 = # filters; 28×28 = # neurons that each filter needs)

(3) Stride difference $= 2$

neurons $= 6 \times (14 \times 14) = 1176$

(6 = # filters; 14×14 = # neurons that each filter needs)

(b)(1) To reduce weights and computational complexity

(2) Pooling reduces computational complexity, shortens image recognition time, improves results and avoids overfitting.

4-(a)(b):

Customer Name	Ground Truth on the Fraud	Testing Scores by Training	Machine Prediction Validation States Results: (Yes or No) (TP, TN, FP, or FN)
John	Yes	0.82	Yes、TP
Bob	No	0.75	Yes、FP
Mary	Yes	0.62	No、FN
Sophie	No	0.45	No、TN

(c) $\text{accuracy} = \frac{TP+TN}{TP+TN+FP+FN} = \frac{1+1}{1+1+1+1} = \frac{1}{2} = 0.5$

(d) Reason: The model deliberately memorizes the disruption properties of the training samples. In other words, the model being created is overly biased by the sample data behaviour. The sample dataset may stay far away from common data distribution or characteristics in general applications.

Methods: ① Increase the training data size. Sometimes, manual labeling is added to generate some artificial training samples. For instance, one can apply image recognition, mirror transformation, and rotation to enlarge sample quantity.

② Use feature screening and dimension reduction methods. We can traverse all combination styles of features and select the more important features.

③ Simplify the model (By reducing # neurons per layer to make the network smaller)

④ Stop early when training, which is a form of regularization while training a model with an iterative method.

Cloud Computing Homework 3 (problem 5-8)

Problem 5

Code for generating the dataset:

- The code in **DataGen.java** under the directory **/HiBench/autogen/src/main/java/HiBench** would be executed first (shown in figure 5.1), invoking the data generation class **BayesData** in **BayesData.java** in the directory **/HiBench/autogen/src/main/java/HiBench** (shown in figure 5.2).

```
8  public class DataGen extends Configured implements Tool {
9
10         public static final boolean DEBUG_MODE = true;
11
12         @Override
13         public int run(String[] args) throws Exception {
14
15                 DataOptions options = new DataOptions(args);
16                 switch (options.getType()) {
17                         case HIVE: {
18                                 HiveData data = new HiveData (options);
19                                 data.generate();
20                                 break;
21                         }
22                         case PAGERANK: {
23                                 PagerankData data = new PagerankData(options);
24                                 data.generate();
25                                 break;
26                         }
27                         case BAYES: {
28                                 BayesData data = new BayesData(options);
29                                 data.generate();
30                                 break;
31                         }
```

Figure 5.1: General data generation class

```
23 public class BayesData {
24
25         private static final Log log = LogFactory.getLog(BayesData.class.getName());
26
27         private DataOptions options;
28         private Dummy dummy;
29         private int cgroups;
30
31         BayesData(DataOptions options) {
32                 this.options = options;
33                 parseArgs(options.getRemainArgs());
34         }
35
36         private void parseArgs(String[] args) {
37
38                 for (int i=0; i<args.length; i++) {
39                         if ("-class".equals(args[i])) {
40                                 cgroups = Integer.parseInt(args[++i]);
41                         } else {
42                                 DataOptions.printUsage("Unknown bayes data arguments -- " + args[i] + "!!!");
43                                 System.exit(-1);
44                         }
45                 }
46         }
47
```

Figure 5.2: Specified data generation class for Bayesian data

- The task contains **12** MapReduce jobs running in series
- Two experiment sets are performed in the Bayesian Classifier training bench:

Environment: Figure 5.3 showing the running time varying with datasize. The definition of "large" datasize is shown in figure 5.4. All settings unmentioned apply the default settings. As shown in figure 5.3, the datasize of small and huge gives too short and long running time for analysis, respectively. As a resutlt, the default datasize for this experiment is selected as large, which has proper execution time and iterations for the analysis of serial jobs and acelerations.

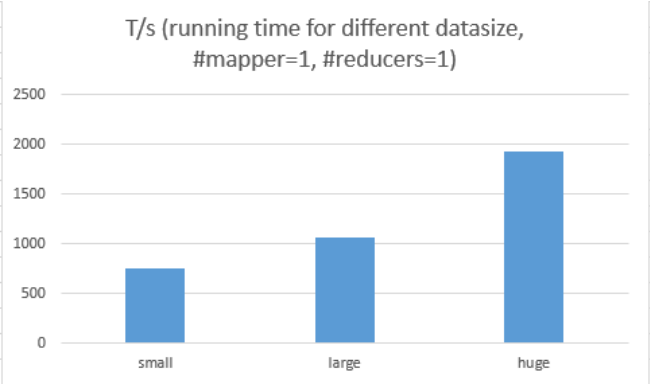


Figure 5.3: Running under different datasizes

```
7   hibench.bayes.large.pages                       100000
8   hibench.bayes.large.classes                     100
9   hibench.bayes.large.ngrams                      2
```

Figure 5.4: Definition of "large" datasize

Definitions:

- *Speedup*: When a cluster is running a particular task, the speedup of the cloud system a (with running time L_a) compared to cloud system b (with running time L_b) in latency is defined by the following formula:

$$S_{(a,b)} = \frac{L_b}{L_a}$$

- *Efficiency*: The cloud efficiency E_a of cluster a containing k types of machines is defined as follows:

$$E_a = \frac{S_{(a,u)}}{(N_a/N_u)}$$

  Where $S_{(a,u)}$ denotes the speedup of a in comparison to u. In this problem u denotes a cluster with two 1-ECU instances (in this problem assuming that each mapper and reducer have 1-ECU computing power). N_a is defined as:

$$N_a = \sum_{i=1}^{k} (n_i \times c_i)$$

  With n_i denoting the number of machines for type-i (which also reflects the maximum speedupthat can be achieved theoretically) and c_i denoting the computing power of each machine instance of type-i. It indicates whether or not the increase in speedup matches the improve in cluster size. $N_a \geq 1$ indicates that such cluster improvement is worthwhile and $N_a < 1$ indicates that the improvement is unworthy compared to the increase in cluster size.

Experiment set A: Fixing # mappers and increasing # reducers:

Procedure: The number of mappers is fixed as 1. I change the number of reducers to a number varying from 1 to 3 and record the running time, speedup and efficiency at every state and then plot them out. After that, I looked deep into the iterations to find out where the speedup and efficiency take place.

Results: The plot of the running time varying with the number of reducers is shown in figure 5.5. The plot of the speedup and efficiency varying with the number of reducers is shown in figure 5.6, 5.7, respectively (u

denotes the cluster with 1 mapper and 1 reducer).

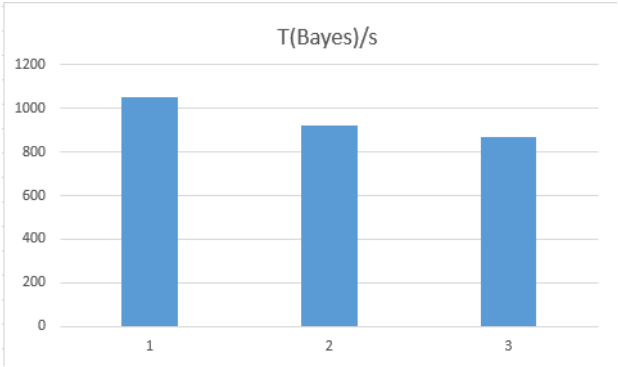


Figure 5.5: Running time under different # reducers

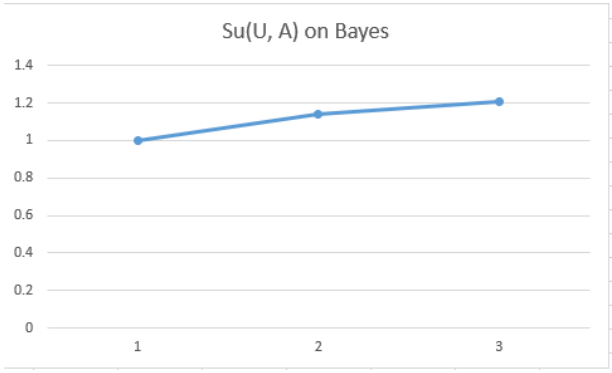


Figure 5.6: Speedup under different # reducers

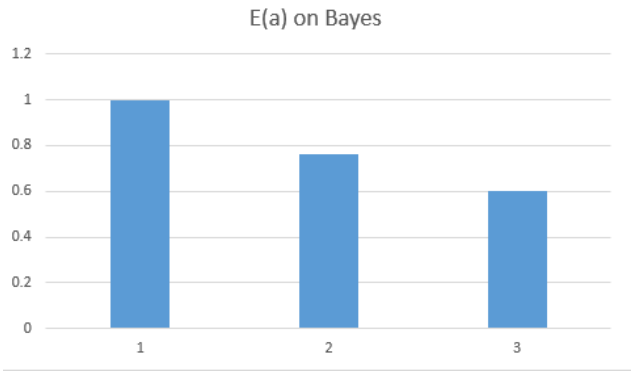


Figure 5.7: Efficiency under different # reducers

Discussions: As shown in figure 5.5, the running time decreases and the speedup increases as the number of reducers used increase, indicating that the use of more reducers improve the performance of training Bayesian classifier. However, we also notice that such improvement is limited in that the efficiency, as shown in figure 5.7, keeps decreasing, indicating that such improvement is unworthy compared to the increase of cluster size. Two possible reasons might explain the limited improvement:

One is **the limited number of serial jobs that are reducer-critical**. Since the training part of Bayesian classifier is divided into multiple stages (the stages for transferring the sequential data into sparse data, for counting and calculating the prior probability, for calculating the conditional probability and for merging the new model with the old model, etc), it is not guaranteed that all jobs involve scaling the number of reducers. As shown in figure 5.8 and 5.9, the number of reducer tasks available can vary due to the jobs' differences. For example, job 1 doesn't require reducer at all and job 6, 7, 9, 11, 12 might have too low degree of parallelism that they cannot be distributed to different reducers. As a result, the time taken to perform such reducer-critical tasks cannot be optimized by solely increasing the number of reducers, making the improvement of speedup limited and the efficiency lower. The variance average processing time per reducer at different jobs,

as shown in figure 5.9, matches the change in the number of reducer tasks launched, hence further verifying the effect of reducer-critical tasks.

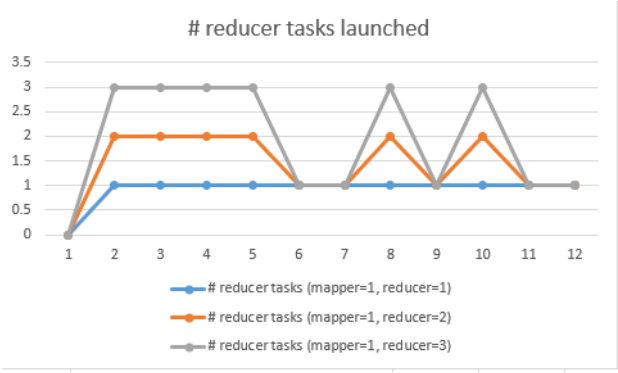


Figure 5.8: # reducer tasks launched under different jobs

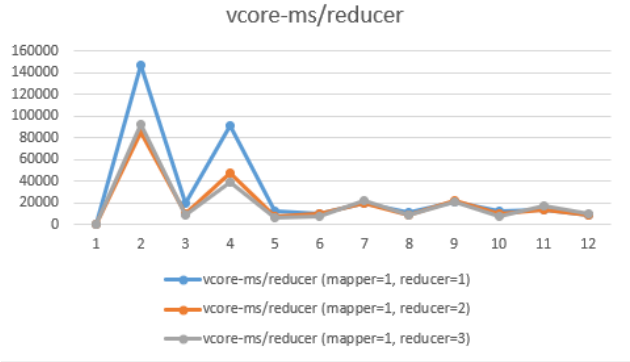


Figure 5.9: Average processing time of a v-core under different jobs

The other reason can be **the increased cost of shuffling** due to the increased reducer number. As shown in figure 5.10, the number of shuffled maps increase when the number of reducers used increase, making the I/O time consumption for shuffling the map outputs to the reducers more heavy.

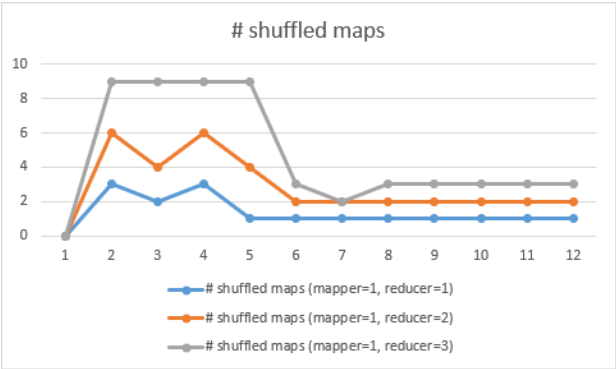


Figure 5.10: # shuffled maps under different jobs

Experiment set B: Fixing # reducers and increasing # mappers:

Procedure: The number of reducers is fixed as 1. I change the number of mappers to a number varying from 1 to 3 and record the running time, speedup and efficiency at every state and then plot them out.

Results: The plot of the running time varying with the number of reducers is shown in figure 5.11. The plot of the speedup and efficiency varying with the number of mappers is shown in figure 5.12, 5.13, respectively (u denotes the cluster with 1 mapper and 1 reducer).

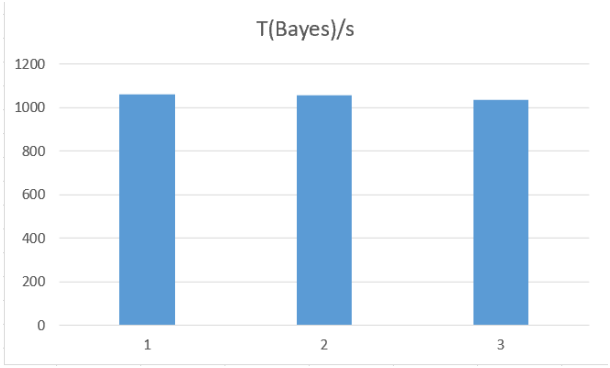


Figure 5.11: Running time under different # reducers

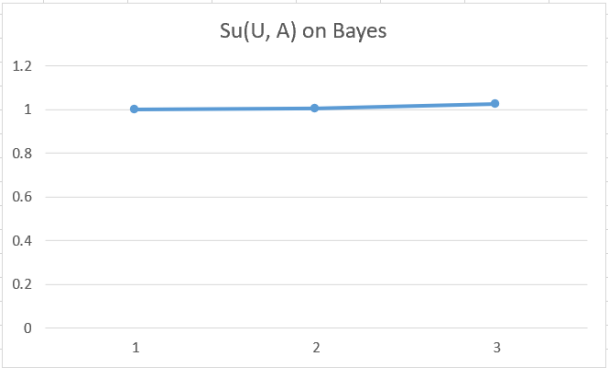


Figure 5.12: Speedup under different # mappers

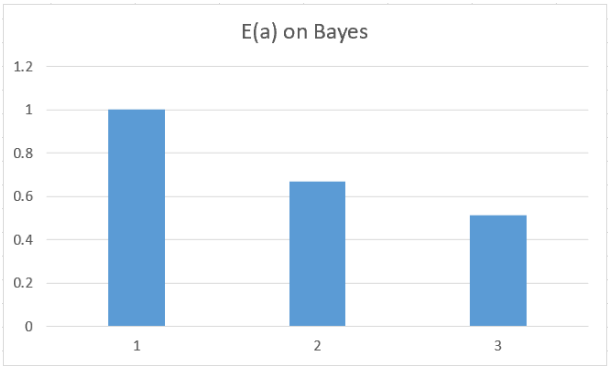


Figure 5.13: Efficiency under different # mappers

Discussions: As shown in the above figures, the speedup didn't vary much with respect to the number of mappers and on the other hand, the running time slightly increases as the number of mappers increase. A possible reason is that **the increase of the number of mappers induces the occurance of more map tasks in some mapper-critical jobs, which makes communication more expensive**. As shown in figure 5.14, the number of mapper tasks may increase for some jobs relying on more mappers. However, the time taken on average for a map task to finish the job didn't vary much, as shown in figure 5.15, indicating that the performance didn't improve a lot even though more map tasks are launched (which is possibly because of the low degree of parallelism nature of the jobs' mapper part). Hence, a more reliable explanation is that more mapper tasks have to be executed concurrently, making it necessary and costly to communicate with each other for synchronization.

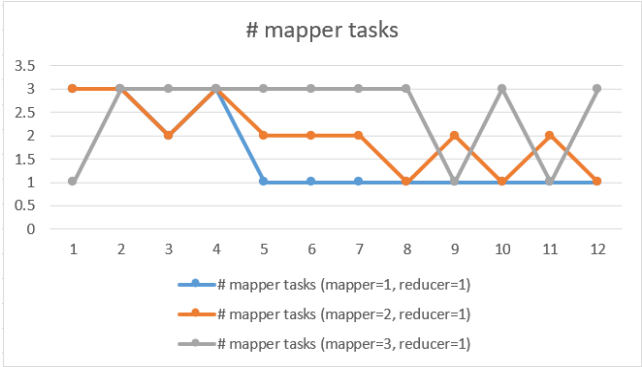


Figure 5.14: # mapper tasks under different jobs

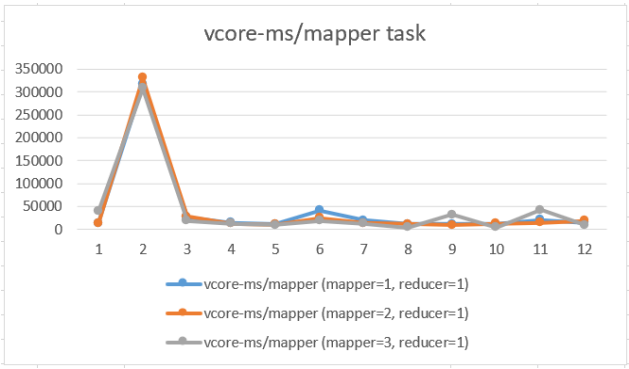


Figure 5.15: vcore-ms/mapper under different jobs

Problem 6

 # classes and features used: 3 classes are used. 4 attributes (features) are used for clustering.

Default experiment: The default setting's (with num_cluster=3) program runs only **2 seconds** with accuracy **0.8933333333333333**. The clustering results are shown in figure 6.1. I notice that the accuracy differs for different running trials. The potential reason is that each time a different group of random centroid seed is used, which makes the classification results differ a lot depending on the whether or not the random centroids coincide with the original data's distribution.

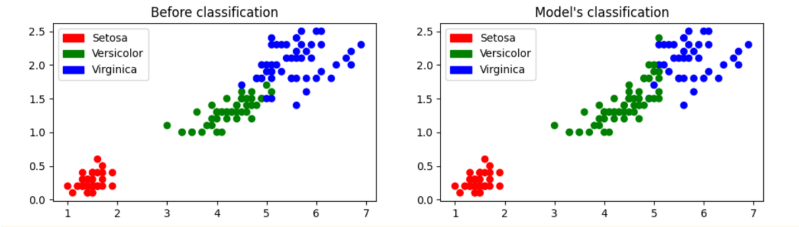


Figure 6.1: Default clustering results

Experiments with different # clusters: The experiments with the number of clusters to be 3, 5, 7 are performed. The change of **running time and accuracy** are shown in figure 6.2. The experimental group settings are shown below (other settings unmentioned remain default):

```
A: num_cluster=3 (Default)
B: num_cluster=5
C: num_cluster=7
```

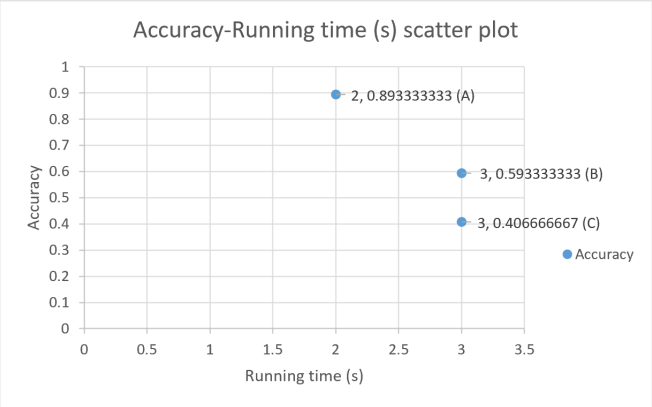


Figure 6.2: Accuracy-Running time scatter plot

The **classification results** of B, C are shown in figure 6.3, 6.4, respectively.

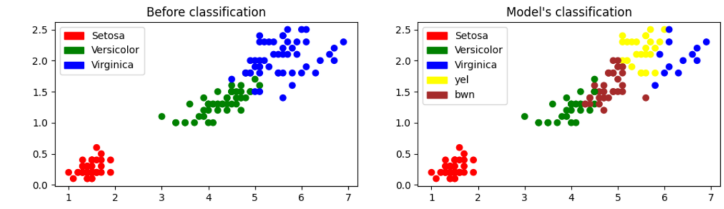


Figure 6.3: Clustering results with num_clusters=5

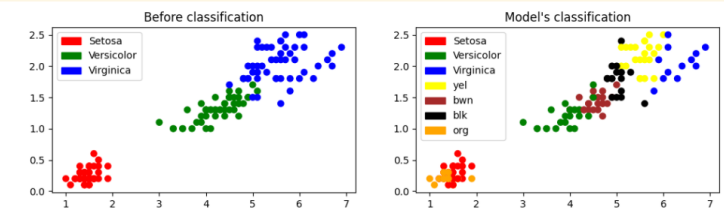


Figure 6.4: Clustering results with num_clusters=7

Due to the limited datasize (150 instances in total with 50 instances per class), the running time didn't vary a lot.

We might notice that the accuracy is decreasing as the number of clusters used increases. Such effect would be a double-sided sword:

On the one hand, it might be **harder for a model developed by larger num_clusters to predict the incoming input correctly** (e.g. predicting a general, less characteristic input labelled 1 as 1), since the actual prediction accuracy decreases.

On the other hand, it **filters the inputs that are less correlated to a particular cluster and preserves those that are characteristic enough to be classified together**. For example, we might notice from figure 6.1, 6.3, 6.4 that for the green cluster, even though the number of inputs that are classified belonging to the green cluster decreases, those classified into green cluster become more characteristic (say, more closed to the green centroid) to be green. In a word, adding more clusters can be regarded as adding more disturbances on classes, which help selecting stronger candidates for a certain desired cluster and filtering those that are not characterized enough on a class.

Hence, such a behavior (adding the number of clusters) can be regarded as **sacrificing accuracy for stronger selection**.

Problem 7

The network used: The original network is LeNet, which is a sort of Convolutional Neural Network (CNN). The structure is shown in figure 7.1.

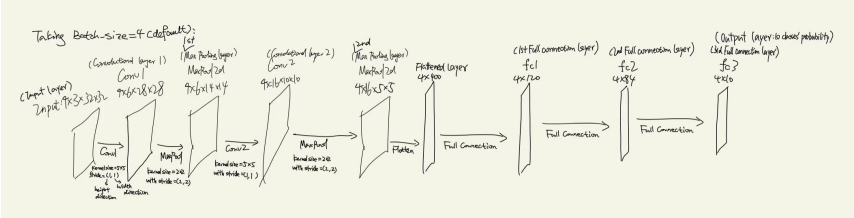


Figure 7.1: Network structure

Default setting running: The experiment with default setting:

```
    Namespace(batch_size_test=4, batch_size_train=4,
dataroot='/group12/cloudComputing2021/DATA/Cifar-10', epochs=5, is_gpu=True,
lr=0.001, momentum=0.9)
```

is run. The training time is around **6 minutes 22 seconds**. The accuracy is about **62%**.

Defeating the default settings: Trials to defeat the defaults settings are shown in figure 7.2. I performed 3 groups of experiments in order to try defeating the default settings:

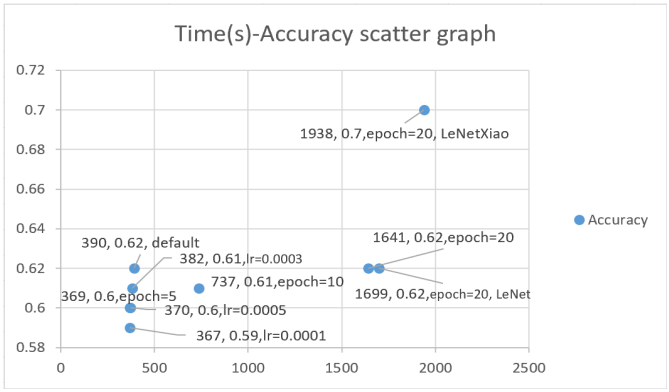


Figure 7.2: Experiments to try defeatin default case

The first group involves **changing the learning rate** with default case and from 0.0001 to 0.0005 with stepsize 0.0002 (fixing epoch=5, batch_size=4, using LeNet). The results show that the accuracy and running didn't vary much with respect to the change of learning rate. This probably is because the loss had converged to the local minimal (due to the excellent default settings given by Shuai Ge) in a suitable enough rate that we don't require a faster or slower convergence.

The second group involves changing **the number of epoches** with 5, 10 and 20. The results show that as the number of epoches increase, the running time increases significantly due to the training costs for more epoches. On the other hand, I noticed that the accuracy improved slightly, which shows that training more epoches did help make the objective value more closed to the local minimal (by taking more gradient moves), even though such descending motion is limited due to the excellent default settings given by Shuai Ge. It inspired me to take larger number of epoches when using my modified network.

The third group involves **modifying the network**. Given fixed epoch=20, lr=0.0003,batch_size=4, the new network LeNetXiao's structure is shown in figure 7.3 (with codes specified in figure 7.4). Specifically I changed the number of output channels (for conv1) and the number of input channels (for conv2) to 24, enabling conv1 to extract more different kinds of features so that the model can be more robust on inputs with

different features. I also changed the second max pooling layer with the average pooling layer, which helps reduce the effect of intense light in the inputs and retains more generic features in the output feature maps. LeNetXiao works possibly because the dataset itself has less features that are intense in light but more features that are general but not fully extracted by the default network.

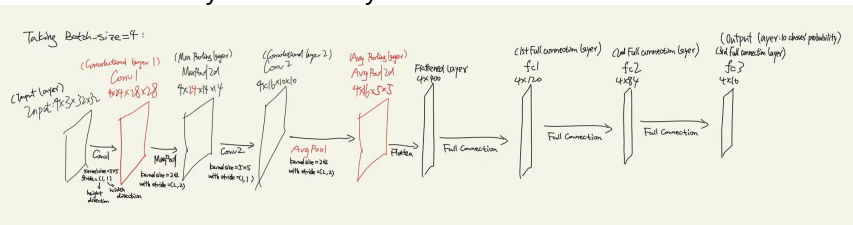


Figure 7.3: LeNetXiao network structure

```python
# max pool + avg pool, middle layer uses 24
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 24, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.avgpool = nn.AvgPool2d(2, 2)
        self.conv2 = nn.Conv2d(24, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.avgpool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

```

Figure 7.4: LeNetXiao network code

Classification results: The classification results for the default network is shown in figure 7.5.

Figure 7.5: default network results

CPU/GPU utilization rate analysis: The CPU and GPU utilization rates varying with respect to time for the default network are shown in figure 7.6, 7.7, respectively. The CPU and GPU utilization rate for LeNetXiao, the best-performance case, are shown in figure 7.8, 7.9, respectively.

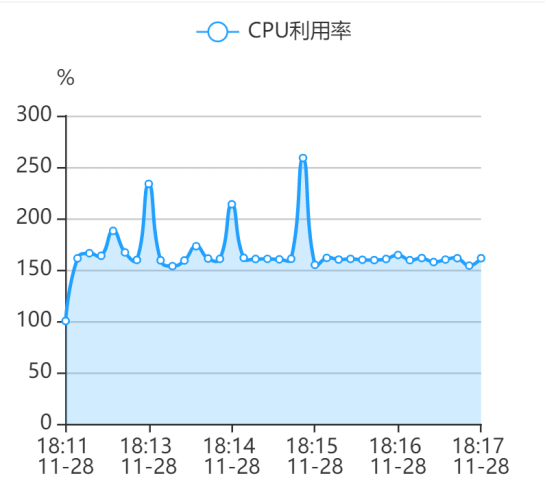


Figure 7.6: Default case: CPU utilization rate w.r.t. time

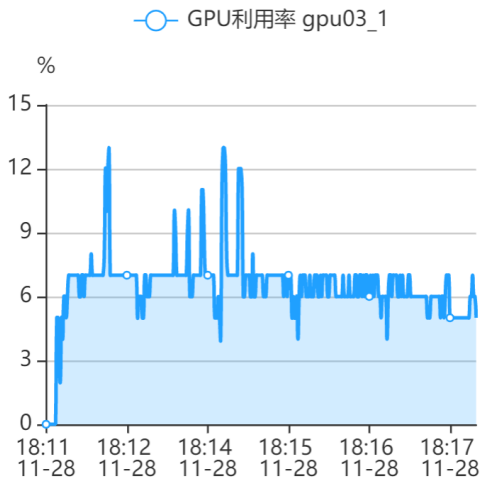


Figure 7.7: Default case: GPU utilization rate w.r.t. time

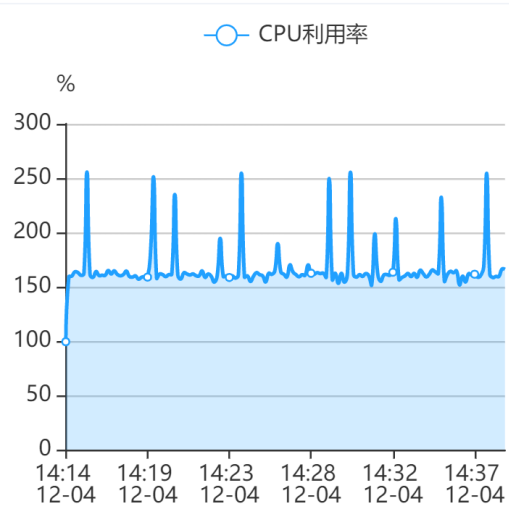


Figure 7.8: Best case: CPU utilization rate w.r.t. time

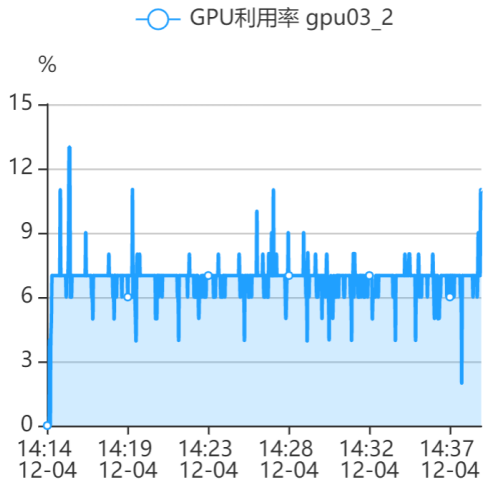


Figure 7.9: Best case: GPU utilization rate w.r.t. time

The performance on GPU utilization is periodically dependent on the **epoch-training behaviors** (high when the network forwarding and backward propagation processes are going on and low at the end of each epoch), as suggested by the number of valleys in the GPU utilization graph (approximately 4 in the default case and 20 in the best case) which coincides with the number of epoches.

The performance on CPU utilization may depend on **the communication behaviors between servers or CPUs and GPU card**. For example, after batches of training the CPU would have to launch more threads to fetch data (new model parameters or output predicted results, etc) from other servers using OpenMP, which increases CPU's load on scheduling and managein those communication behaviors.

I also noticed that **the average (and maximum) of CPU and GPU utilization rate didn't vary much** for the best case and the default case. This is possibly because the batch size (set as 4) is unchanged and the network structure didn't change significantly (no layer is added and just some layers are modified), which didn't make the (GPU and CPU) memory consumption vary much.

Problem 8

Perplexity: perplexity is a measurement of how well a probability distribution or probability model predicts a sample. It is defined as the exponential of the cross entropy loss:

$$P(x) = e^{H(x)} = e^{-\sum_x p(x) \times log_e p(x)}$$

Perplexity can be interpreted as the "number of choices" the random variable has. For example, $P(x) = 7$ for the current model indicates that there are 7 most probable (having the highest and closed prediction probability) choices for the text of image x.

Training time, final perplexity, loss and optimized results: The experiments I performed are divided into 6 groups with the following settings (letter representing experiment index and the content representing the settings. Settings that are not mentioned are kept with default values):

```
A:  batch_size=256, embedded_size=299, hidden_size=256
B:  batch_size=128, embedded_size=299, hidden_size=256 (Default)
C:  batch_size=128, embedded_size=299, hidden_size=1024
D:  batch_size=128, embedded_size=149, hidden_size=2048 (Best)
```

```
E:  batch_size=128, embedded_size=299, hidden_size=2048
F:  batch_size=128, embedded_size=598, hidden_size=2048
```

The training times under different settings are shown in figure 8.1. The final perplexity and loss of the model predictions are shown in figure 8.2.

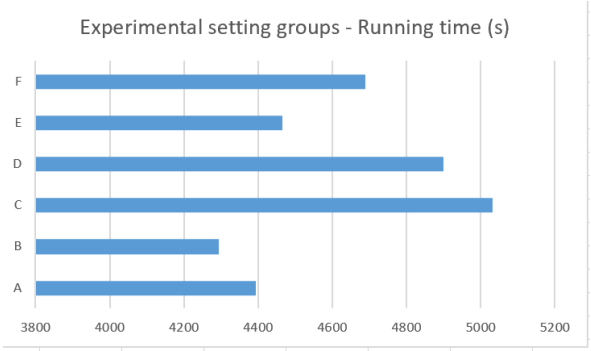


Figure 8.1: Training time under different settings

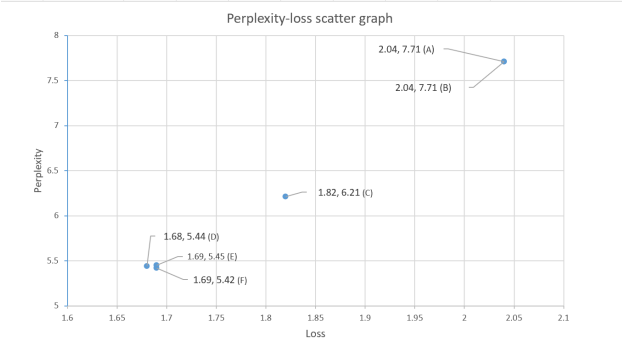


Figure 8.2: Final perplexity and loss under different settings

Comparing the results obtained from experiments with settings A and B, I found that **the change of batch size** didn't affect the running time, loss and perplexity a lot. It indicates that the default batch size can provide a suitable descend direction for reducing the loss as well as enable an acceptable training speed.

Comparing the results from experiments with settings B, C and E we figure out that the larger the **hidden size**, the less the loss and perplexity would be. The potential reason is that the increase of hidden size increases the utilization rate of the hidden states on the features given from the encoder outputs, which may make the model more comprehensive at predicting new incoming inputs (by evaluating more features).

The comparisons of the results from experiments with settings D, E and F we found that the loss and perplexity didn't vary much even though different **embedded size** is used. This may due to the constant hidden size, which ensures that the prediction is based on enough faetures and no more modifications in the embedded layer is required. Such observation is meaningful in that we can reduce the embedding size to 149, which reduces the device memory load and possible I/O costs, without hurting the performance. As a result, experiment set E is selected as the best setting group in that it had the best performance on perplexity and loss, as well as requires less loads on memory.

Changes in perplexity and loss for a given setting: The changes of loss for the default and the best group are shown in figure 8.3. The changes of perplexity correspondingly are shown in figuer8.4.

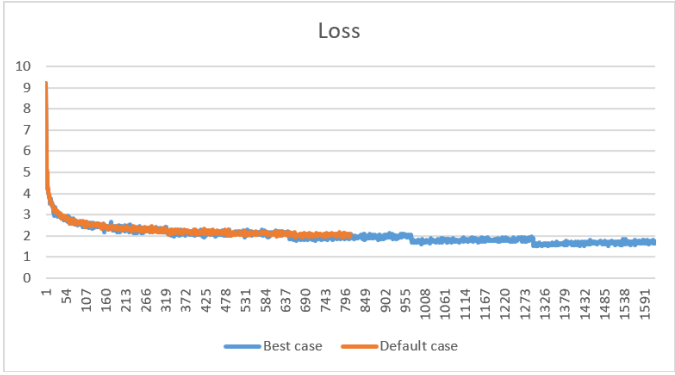


Figure 8.3: The change of (cross entropy) loss with respect to logging steps

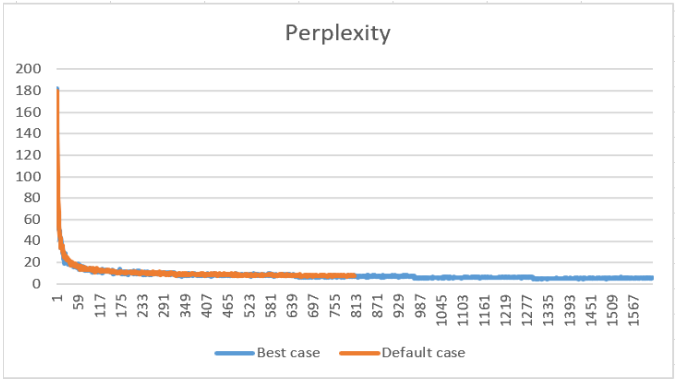


Figure 8.4: The change of perplexity with respect to logging steps

We can notice that the slope for both curves are closed to each other, indicating that actually both cases share similar instantaneous convergent speed. However, since the batch size is fixed (taking more time and reducing more), we may infer that the reducing effect on loss and perplexity for each batch for the best case is better than that of the default case (say, training 1 batch in the best case worths training 2 batches in the default case). Such potential reason is reliable in that it explains the fact that the decrease of loss and perplexity under the best case is more permanent: batch trainings are more permanent under the best case.

CPU/GPU utilization analysis: The CPU and GPU utilization rates for the default case are shown in figure 8.5, 8.6, respectively. The rates for the best case are shown in figure 8.7, 8.8.

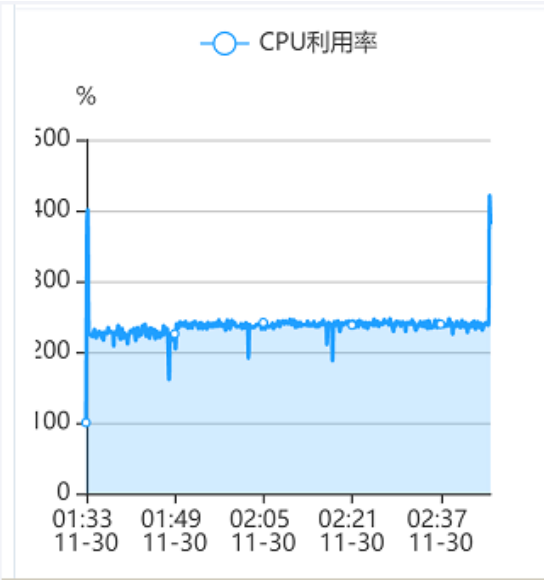


Figure 8.5: Default case: CPU utilziation rate

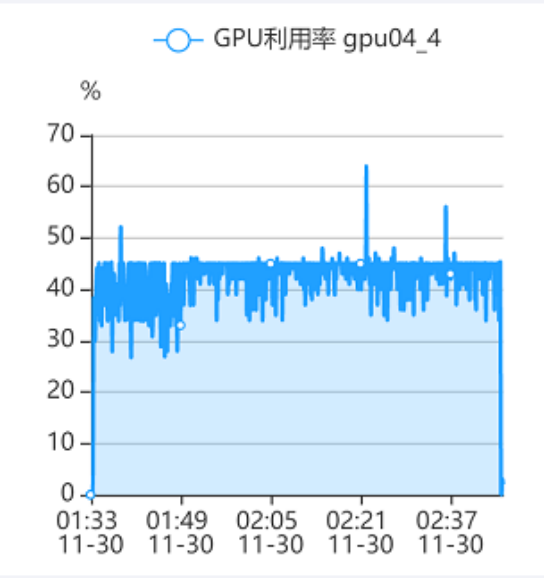


Figure 8.6: Default case: GPU utilziation rate

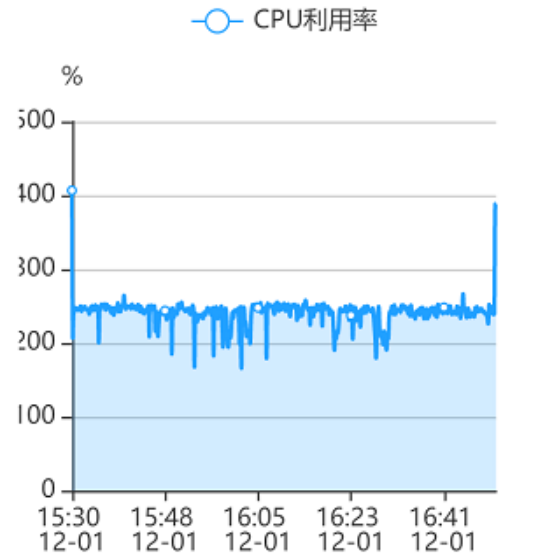


Figure 8.7: Best case: CPU utilziation rate

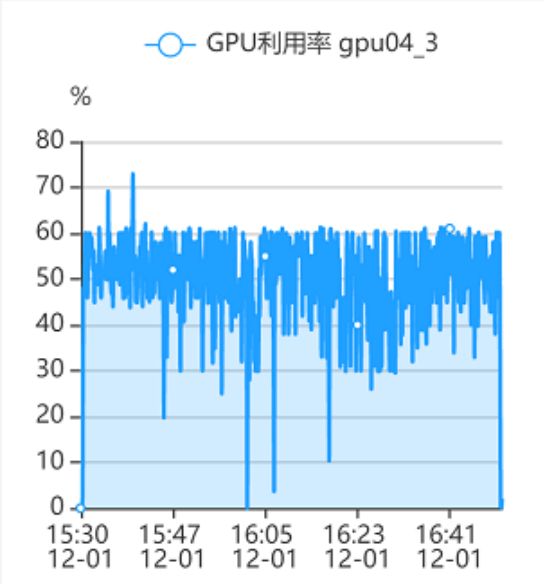


Figure 8.8: Best case: GPU utilziation rate

We can notice that under the default case, **the CPU utilization** rates didn't vary much, indicating that the communication cost (supplying data for device or other servers) is relatively low and that most time are spent on GPU encoding and decoding the batched data. **The GPU utilization** rates under the default case swings corresponding to the beginning of batches and epoches, indicating that there might be some GPU performance costs at switching from one batch or epoch to the next.

We may also notice that under the best case, **the behavior of the CPU utilization rate is similar to that under the default case** (which can also be explained by the scarcity of communication costs), but **the overall GPU utilization rate is much higher (on about 10%) than that under the default case**. A potential reason is that larger hidden size results in more intermediate feature-relevant data in RNN, which requires more computing power to generate those faeture vectors and interpret those vectors into output texts.