Light-ffm

Jiwei liu

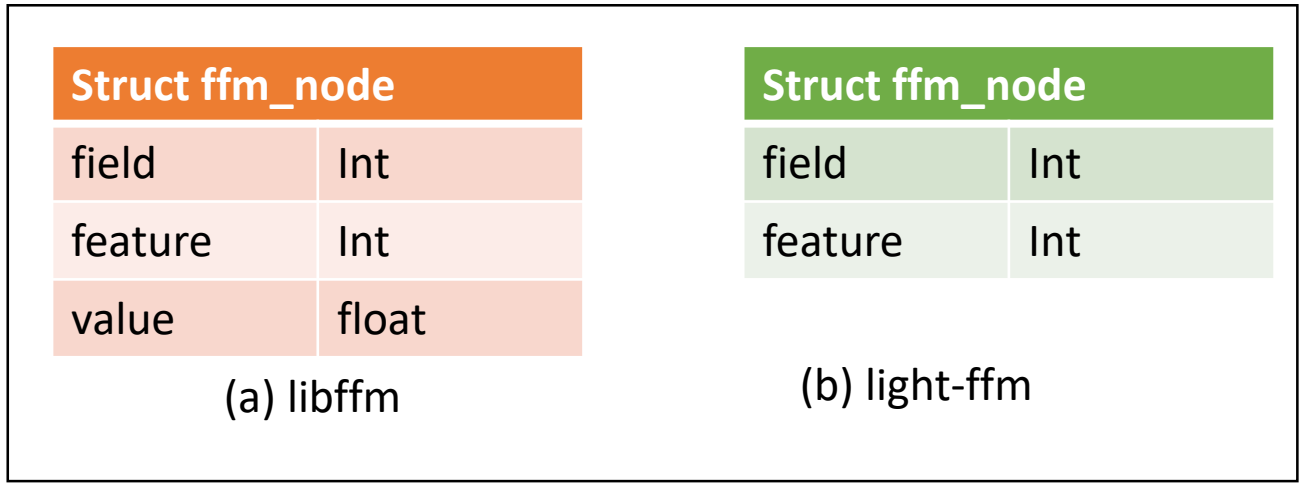
https://github.com/daxiongshu/light-ffm

Overview

- Design
  - A faster online mode with data caching
  - Field remap matrix to reduce memory usage and make feature selection possible without rewriting the data file.
  - Per group softmax cross entropy cost function to improve apk. It is also faster to converge.
- Usage
  - Data Format
  - Command line arguments
- QA
- Bugs & issues

The memory usage of ffm: data part

- The memory usage of libffm includes two parts: data + weights(latent vectors). Let's look at data first



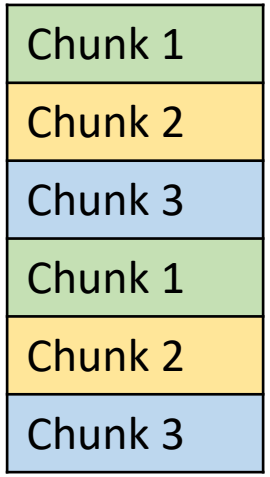
Data structure in ffm.h

- In practice, we found ffm is much better with categorical feature, where value=1, than numerical features where value can be any real number.
- Therefore, we don't store value at all since it is always 1. This can cut 33% of memory usage of data part of ffm.

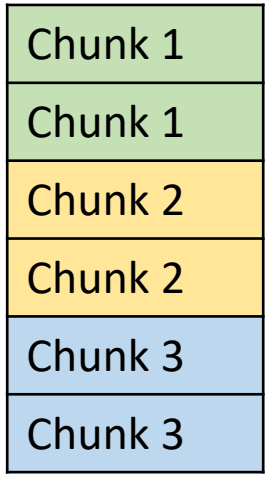
The memory usage of ffm: data part

- ffm is a totally online algorithm, which means the weights are updated per sample/row of data. Hence there is no reason to store entire data in memory all at once. We can read data chunk by chunk and learn weights with one chunk at a time.

(a) Train data is split in a interleaved way so that any single chunk has rows from all timestamps. In each iteration/epoch, chunks are shuffled and within each chunk samples are also shuffled



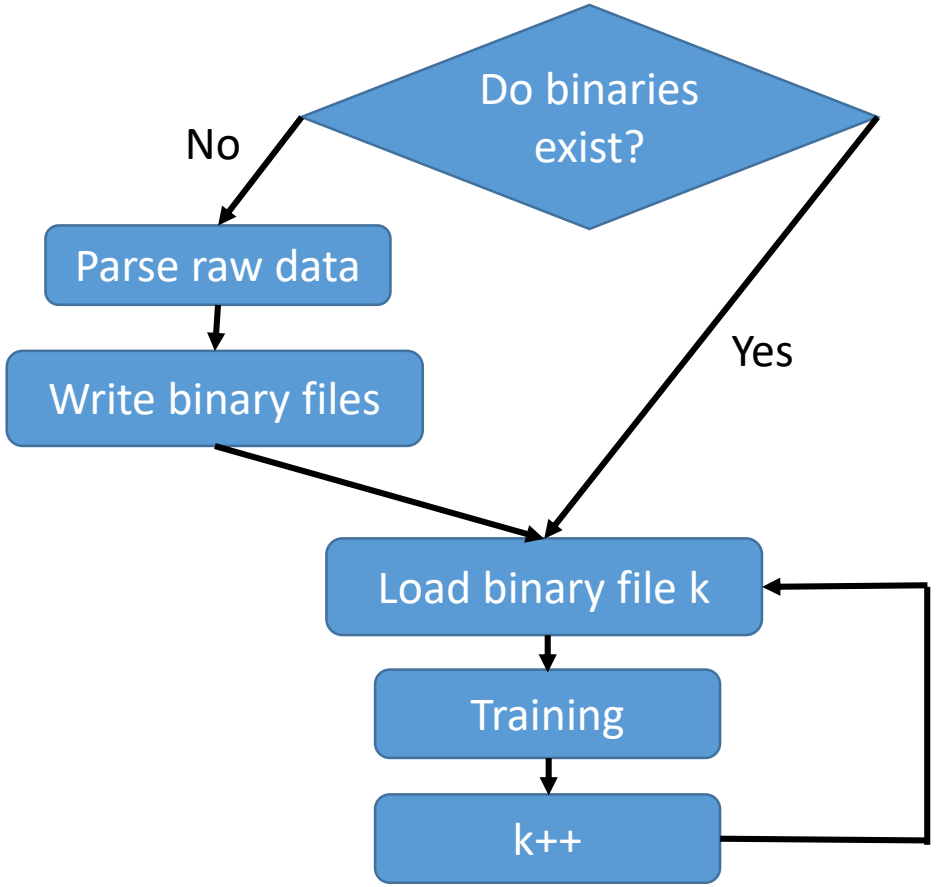
(b) Test data is split in a continuous way and within each chunk samples are not shuffled so that the test file row order is preserved.



- Each chunk is stored in c++ binary format which is fast for reloading. For example, if the input data are train.fm and test.fm, you will find train.fm.0.b ~ train.fm.9.b and test.fm.0.b and test.fm.9.b files generated in the same folder. The *.b files are binary chunks. By default, data are split into 10 chunks so we reduce data memory footprint by 90%.

Data caching.

- Raw input data is parsed only in the first time. All following training will load binaries directly, which is much faster.



Separate context feature from AD feature

- We further observe that there are redundancy in default ffm data format.
- For example, for a display_id, we use two features, document_id and platform. For Ads, we use two features, ad_id and publisher. Assume this display_id has 4 Ads and fields mapping are document_id:0, platform:1, ad_id:2, publisher:3. Then default ffm may have data look like this:

(a) Data format in default ffm. Display_id/context features are duplicated for all rows

```
1 0:0:1 1:1:1 2:2:1 3:3:1
0 0:0:1 1:1:1 2:4:1 3:5:1
0 0:0:1 1:1:1 2:2:1 3:3:1
0 0:0:1 1:1:1 2:4:1 3:5:1
```

(b) Data format in light-ffm. Display_id/context features written in the first row start with ‘x’

```
x 0:0:1 1:1:1
1 2:2:1 3:3:1
0 2:4:1 3:5:1
0 2:2:1 3:3:1
0 2:4:1 3:5:1
```

- As shown in (b), each display group starts with a ‘x’ row of common context features by each AD.

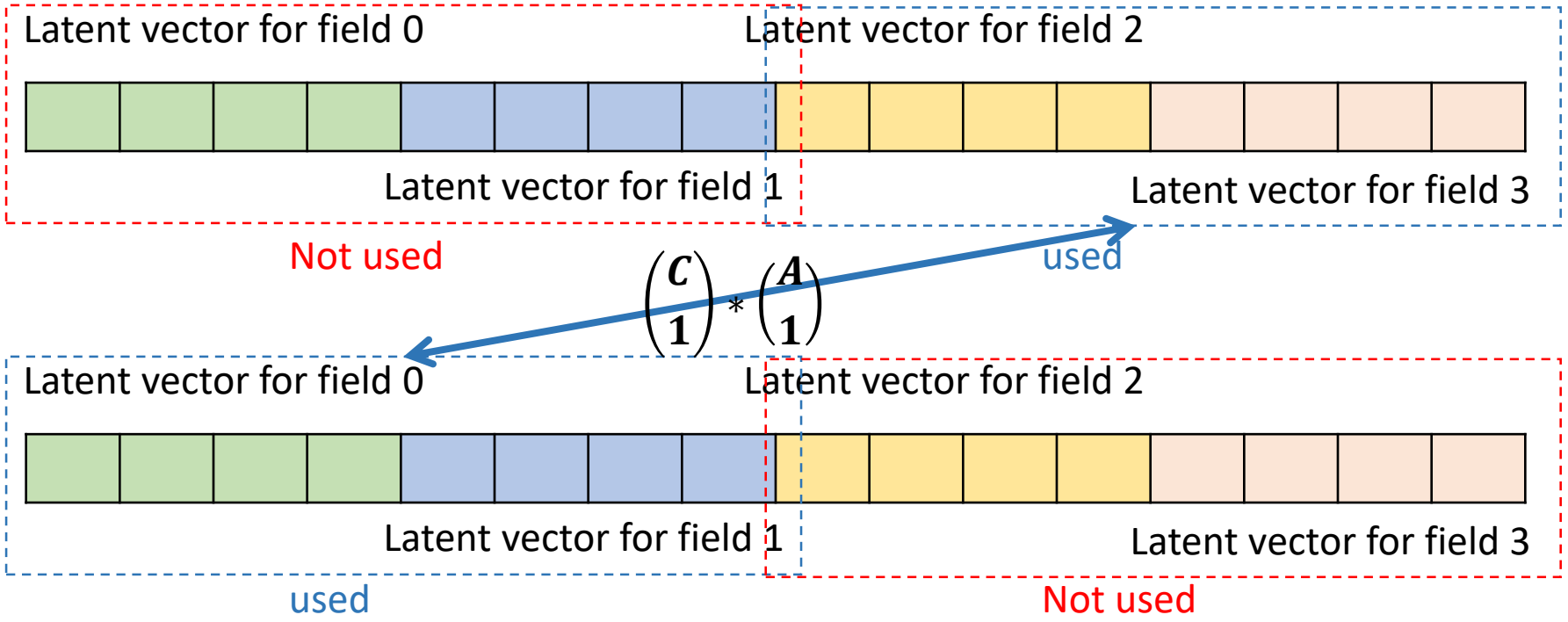
Observation: only keep strong interactions.

- FFM by default uses full-2way interactions. Assuming M fields, $\binom{M}{2}$ inner products of interactions will be summed to get the final prediction.
- In case like outbrain, we can divide M fields into two types: M = C + A, where C is the number of context fields and A is the number of AD fields. So we have $\binom{M}{2} = \binom{C}{2} + \binom{A}{2} + \binom{C}{1} * \binom{A}{1}$ where $\binom{C}{2}$ interaction between two context fields, $\binom{A}{2}$ means interaction between two AD fields. $\binom{C}{1} * \binom{A}{1}$ means interaction between one context field and one AD field. It is obvious that $\binom{C}{2}$ doesn't affect apk since it is the same for all Ads . In practice we also find $\binom{A}{2}$ weak**. So we could just calculate $\binom{C}{1} * \binom{A}{1}$ to achieve almost the same performance as $\binom{M}{2}$, which could be much cheaper/faster than $\binom{M}{2}$.**

Unused latent vectors for $\binom{C}{1} * \binom{A}{1}$ only interactions

Context fields: document_id:0, platform:1
AD fields: ad_id:2, publisher:3

Latent vector of document_id, field 0, a context field



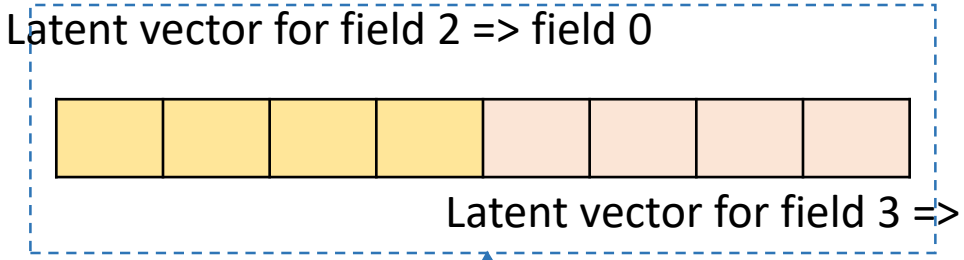
Latent vector of ad_id, field 2, an AD field

- It is obvious some latent vectors of some fields are not used at all if we only consider $\binom{C}{1} * \binom{A}{1}$ type interactions.

Remapping fields with $\binom{C}{1} * \binom{A}{1}$ only interactions

Context fields: document_id:0, platform:1
AD fields: ad_id:2, publisher:3

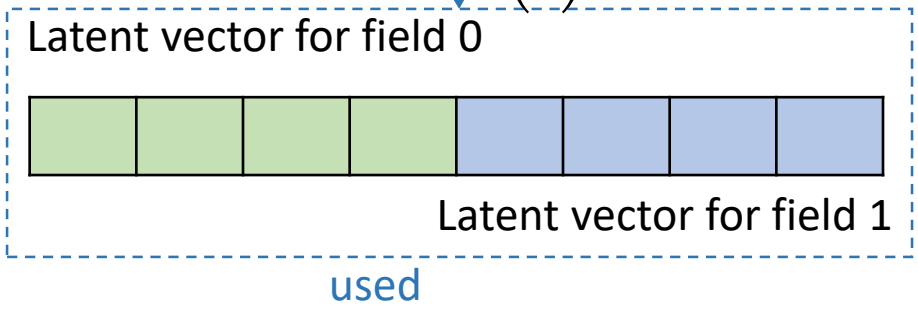
Latent vector of document_id, field 0, a context field



$\binom{C}{1} * \binom{A}{1}$

For context fields we can remap field 2 to field 0 and field 3 to field 1 since field 0 and 1 latent vectors are not used anyway.

Latent vector of ad_id, field 2, an AD field



So for AD fields we just throw away field2 and field3 latent vectors.

We can have total number of fields, M=2, instead of 4 if we only consider $\binom{C}{1} * \binom{A}{1}$ type interactions.

Field remap matrix

- light-ffm requires an input file that specifies field reamp matrix as below.

	Field 0	Field 1	Field 2	Field 3
Field 0	-1	-1	0	1
Field 1	-1	-1	0	1
Field 2	0	1	-1	-1
Field 3	0	1	-1	-1

Latent vectors of field 0,
remap field 2 to field 0 and field 3 to field 1

-1 means these interaction are turned off

So now we need only 2 fields instead of 4 fields. In light-ffm, a command line arg “-savefield 2“ must be given so specify that after field remapping, the number of fields is 2.

Field remap matrix

	Field 0	Field 1	Field 2	Field 3
Field 0	-1	-1	0	1
Field 1	-1	-1	0	1
Field 2	0	1	-1	-1
Field 3	0	1	-1	-1

Latent vectors of field 0, **remap field 2 to field 0 and field 3 to field 1**

-1 means these interaction are turned off

In light-ffm, this matrix needs to be provided in following format in a separate txt file:

0: -1 -1 0 1

1: -1 -1 0 1

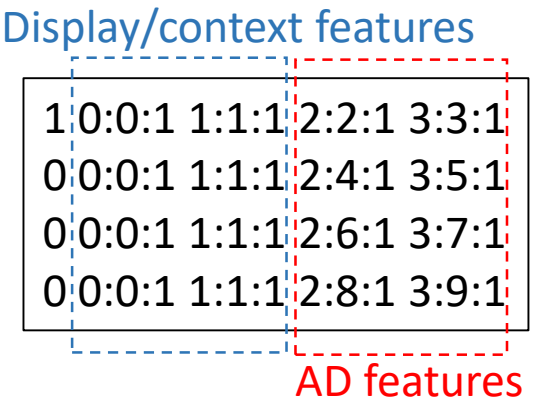
2: 0 1 -1 -1

3: 0 1 -1 -1

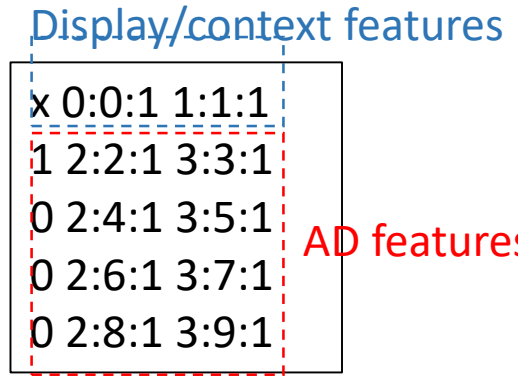
Usage: Data Format

- light-ffm requires a different data format from libffm.

(a) Data format in default ffm. Display_id/context features are duplicated for all rows



(b) Data format in light-ffm. Display_id/context features written in the first row start with 'x'



As shown in (b), each display group starts with a 'x' (lower case x) row of common context features by each AD. `x` indicates this row is for common context features. It is recommended but not required that AD fields and context fields are continuous, for example, AD fields 0~9 and context fields 10~20

Usage command line arguement

../../../myml/light-ffm/ffm-train-predict --auto-stop -savefield 13 --hasLabel -fmap feamap.txt --trainbatch -t 30 -l 0.00009 -r 0.05 -decay 0.02 -s 8 -k 40 -p cvdata/va_sample.fm -test cvdata/va.fm -out cvf.csv cvdata/tr.fm cv-model

Unlike libffm which has ffm-train and ffm-predict, light-ffm has only one executable, ffm-train-predict.
The general format is:
./ffm-train-predict <command line args> train_data.fm model_name

Usage command line arguement

../../../myml/light-ffm/ffm-train-predict --auto-stop -savefield 13 --hasLabel -fmap feamap.txt --trainbatch -t 30 -l 0.00009 -r 0.05 -decay 0.02 -s 8 -k 40 -p cvdata/vaf_sample.fm -test cvdata/va.fm -out cvf.csv cvdata/tr.fm cv-model

These arguments are same with libffm:
--auto-stop, -t, -l, -r, -s, -k
These are new arguments:
--hasLabel: write true labels in output prediction, for cv
-savefield: the total number of fields after field remapping (see page 7~11)
-fmap: file name of field remap matrix (see page 7~11)
-decay: exponential learning rate decay.
-p: file name of validated data
-test: file name of test data
-out: file name of prediction file, output

-savefield, -fmap, -test, -out must be provided!

Validation data

../../../myml/light-ffm/ffm-train-predict --auto-stop -savefield 13 --hasLabel -fmap feamap.txt --trainbatch -t 30 -l 0.00009 -r 0.05 -decay 0.02 -s 8 -k 40 -p cvdata/va_sample.fm -test cvdata/va.fm -out cvf.csv cvdata/tr.fm cv-model

Va_sample.fm is a sampled version of va.fm for both speed and accuracy, please look at https://github.com/daxiongshu/kaggle-outbrain/blob/master/carl/cv_0.6944/sample_va.py

Logs

model initiated, m=13, n=1573467

iter	tr_logloss	va_logloss	va_apk	Time	
1	0.24173	1.45664	0.67167	42.1s	# this is the time for one chunk, so 421s is the real time
2	0.23909	1.35093	0.67527	42.5s	# tr_logloss is the per sample logloss, va_logloss is the per group cross entropy
3	0.23699	1.29896	0.67704	44.3s	# usually we just care about va_apk. Since we use va_sample for online check,
4	0.23553	1.26801	0.67817	44.3s	# this apk is about 0.001 lower than real final apk.
5	0.23382	1.24356	0.67890	44.1s	

QA

1. What are the *.b files?
   They are automatically generated cache files for fast reloading.
2. When I tune parameters like –r, -l, -k, do I need to delete *.b?
   No.
3. When I change –fmap, field remap matrix, do I need to delete *.b?
   No.
4. When I change train.fm, do I need to delete train.fm*b?
   Yes!
5. Are the output prediction the same row order as input test file?
   Yes!