


Imports

```

1 !pip install numerapi
2
3 import matplotlib
4 import numpy as np
5 import pandas as pd
6 from numerapi import NumerAPI
7 import random
8 import sklearn
9 import time
10 import lightgbm
11 import matplotlib.pyplot as plt
12 import gc
13
14 from numba import cuda, jit
15 import numba
16
17
18
19 %matplotlib inline
20

```

 Requirement already satisfied: numerapi in /usr/local/lib/python3.10/dist-packages (2.19.1)
 Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from numerapi) (2.32.3)
 Requirement already satisfied: pytz in /usr/local/lib/python3.10/dist-packages (from numerapi) (2024.2)
 Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from numerapi) (2.8.2)
 Requirement already satisfied: tqdm>=4.29.1 in /usr/local/lib/python3.10/dist-packages (from numerapi) (4.66.5)
 Requirement already satisfied: click>=7.0 in /usr/local/lib/python3.10/dist-packages (from numerapi) (8.1.7)
 Requirement already satisfied: pandas>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from numerapi) (2.2.2)
 Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.0->numerapi) (1.26
 Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.0->numerapi) (202
 Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil->numerapi) (1.16.0)
 Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->numerapi) (2.2.2)
 Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->numerapi) (3.10)
 Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->numerapi) (2.2.
 Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->numerapi) (2024
 /usr/local/lib/python3.10/dist-packages/dask/dataframe/__init__.py:42: FutureWarning:
 Dask dataframe query planning is disabled because dask-expr is not installed.

 You can install it with `pip install dask[dataframe]` or `conda install dask`.
 This will raise in a future version.

 warnings.warn(msg, FutureWarning)

Downloads

```

1 your_public_id = ""
2 your_secret_key = ""
3 your_model_slot_name = ""
4
5
6
7 napi = NumerAPI(your_public_id, your_secret_key)
8
9 current_round = napi.get_current_round()
10
11 napi.download_dataset('v5.0/features.json')
12 napi.download_dataset('v5.0/train.parquet')
13 napi.download_dataset('v5.0/validation.parquet')

```

```
14 napi.download_dataset('v5.0/validation_example_preds.parquet')
```



▼ Data

```
1 import json
2
3 ff = json.load(open('v5.0/features.json', 'rb'))
4
5
6 fs = ff['feature_sets']['all']
7
8
9 df_train = pd.read_parquet('v5.0/train.parquet')
10
11 df_train['era'] = df_train['era'].astype(int)
12
13
14 df_train = df_train[df_train.era < df_train.era.max() - 4]
15
16 Xt = df_train[fs].fillna(2).astype(np.uint8).values
17 Yt = df_train['target'].fillna(0).values
18 Yt = 2*Yt - 1
19 Et = df_train['era'].values
20
21 del df_train
22 gc.collect()
23
24 df_valid = pd.read_parquet('v5.0/validation.parquet')
25
26 Xv = df_valid[fs].fillna(2).astype(np.uint8).values
27 Yv = df_valid['target'].fillna(0).values
28 Yv = 2*Yv - 1
29 Ev = df_valid['era'].values
30
31
32
33 df_sub = df_valid[[]]
34
35 del df_valid
36 gc.collect()
37
38
```

0

▼ Cuda Model Definition

```
1
2
3 class ExtraFastBooster(object):
4
5     def __init__(self, trees_per_layer=8, max_depth=6, nfeatsets=8, lr=.01, L2=10_000,
6
7         self.L2 = L2
8         self.lr = lr
```

```

9     self.max_depth = max_depth
10    self.nfeatsets = nfeatsets
11    self.nfolds = trees_per_layer
12
13
14    qgrad_mbits = qgrad_bits - 1
15
16
17    if max_depth > 8:
18        packed_tree_dtype = np.uint64
19    elif max_depth > 6:
20        packed_tree_dtype = np.uint32
21    else:
22        packed_tree_dtype = np.uint16
23
24    if max_depth > 8 :
25        tree_dtype = np.uint16
26    else:
27        tree_dtype = np.uint8
28
29
30
31    grad_dtype          = ( np.int16 if qgrad_mbits>7 else np.int8   )
32    grad_mbits          = (          15 if qgrad_mbits>7 else        7   )
33    ymax_lg2            = 30
34
35
36
37    residual_sm_a100 = 15 - 2 - 6 - max_depth
38
39
40    assert( residual_sm_a100 >= 0 )
41
42
43
44    hist_warps_per_block_lg2 = max( 4 - max( residual_sm_a100, 0), 0)
45
46
47
48
49    self.grad_dtype = grad_dtype
50    self.tree_dtype = tree_dtype
51    self.packed_tree_dtype = packed_tree_dtype
52    self.hist_warps_per_block = 1 << hist_warps_per_block_lg2
53
54
55
56    @cuda.jit
57    def _et_sample_lb( X, XS, F, tree_set, stride, ):
58
59        f0 = cuda.blockIdx.x
60        bi = cuda.blockIdx.y
61        wi = cuda.threadIdx.x
62
63
64
65        fs = cuda.shared.array(shape=32, dtype=np.uint32)
66        sm = cuda.shared.array(shape=( 32, 32 ), dtype=np.uint32)

```

```

67
68
69     fs[wi] = F[tree_set, 32*f0 + wi]
70
71     cuda.syncwarp()
72
73
74     for i in range(stride):
75
76
77         i_in = 32*stride*bi + 32*i + wi
78
79
80         if 32*stride*bi + 32*i < X.shape[1]:
81
82
83
84             for k in range(4):
85                 ff0 = fs[8*k + 0]
86                 ff1 = fs[8*k + 1]
87                 ff2 = fs[8*k + 2]
88                 ff3 = fs[8*k + 3]
89                 ff4 = fs[8*k + 4]
90                 ff5 = fs[8*k + 5]
91                 ff6 = fs[8*k + 6]
92                 ff7 = fs[8*k + 7]
93
94                 kk0 = (wi + 8*k + 0)%32
95                 kk1 = (wi + 8*k + 1)%32
96                 kk2 = (wi + 8*k + 2)%32
97                 kk3 = (wi + 8*k + 3)%32
98                 kk4 = (wi + 8*k + 4)%32
99                 kk5 = (wi + 8*k + 5)%32
100                 kk6 = (wi + 8*k + 6)%32
101                 kk7 = (wi + 8*k + 7)%32
102
103                 v0 = ( X[ff0, i_in] if i_in < X.shape[1] else 0 )
104                 v1 = ( X[ff1, i_in] if i_in < X.shape[1] else 0 )
105                 v2 = ( X[ff2, i_in] if i_in < X.shape[1] else 0 )
106                 v3 = ( X[ff3, i_in] if i_in < X.shape[1] else 0 )
107                 v4 = ( X[ff4, i_in] if i_in < X.shape[1] else 0 )
108                 v5 = ( X[ff5, i_in] if i_in < X.shape[1] else 0 )
109                 v6 = ( X[ff6, i_in] if i_in < X.shape[1] else 0 )
110                 v7 = ( X[ff7, i_in] if i_in < X.shape[1] else 0 )
111
112                 sm[8*k+0, kk0] = v0
113                 sm[8*k+1, kk1] = v1
114                 sm[8*k+2, kk2] = v2
115                 sm[8*k+3, kk3] = v3
116                 sm[8*k+4, kk4] = v4
117                 sm[8*k+5, kk5] = v5
118                 sm[8*k+6, kk6] = v6
119                 sm[8*k+7, kk7] = v7
120
121
122
123         cuda.syncwarp()
124

```

```

125         for k in range(32):
126             i_out = 32*32*stride*bi + 32*32*i + 32*k + wi
127
128             kk = (k + wi)%32
129
130             XS[f0, i_out] = sm[wi, kk]
131
132
133
134
135
136
137
138
139 @cuda.jit
140 def _prep_vars( L, LE, Y, P, G, stride ):
141
142     ti = cuda.blockIdx.x
143     wi = cuda.threadIdx.x
144
145
146     for i in range( stride ):
147
148         j = 32*stride*ti + 32*i + wi
149
150
151         if j < Y.shape[0]:
152
153
154             for k in range(LE.shape[0]):
155
156                 v = packed_tree_dtype(0)
157                 for d in range(1, max_depth):
158                     v |= L[ k, d-1, j] << ( ( d*(d-1) )//2 )
159
160                 LE[k, j] = v
161
162
163
164                 g = ( np.int64(Y[j]) - np.int64(P[j]) ) >> ( 31 - qgrad_mbits )
165
166                 g = min( max(g, - ( 1 << grad_mbits ) + 1 ), ( 1 << grad_mbits) - 1 )
167
168                 G[j] = grad_dtype( g )
169
170     return
171
172
173
174
175 fstmem = ( 8, max_depth-1, 32 )
176
177
178 @cuda.jit
179 def _repack_trees_for_features( FST, LE, LF, tree_set, stride ):
180
181     fi = cuda.blockIdx.x
182     ti = cuda.blockIdx.y

```

```

183     wi = cuda.threadIdx.x
184
185     fst = cuda.shared.array( shape=fstmem,      dtype=np.uint32)
186
187     trees = cuda.shared.array( shape=(20, 32), dtype= packed_tree_dtype )
188
189
190     for k in range(8):
191         if 8*fi + k < FST.shape[1]:
192             for d in range(1, FST.shape[2]):
193                 fst[k, d-1, wi] = FST[tree_set, 8*fi + k, d]
194
195
196
197     for i in range( stride ):
198
199         j = 32*stride*ti + 32*i + wi
200
201         if j < LF.shape[1]:
202
203
204             for k in range(LE.shape[0]):
205                 trees[k,wi] = LE[k, j]
206
207             for k in range(8):
208
209                 if 8*fi + k < LF.shape[0]:
210                     v = 0
211                     for d in range(1, FST.shape[2]):
212                         v |= trees[ fst[k, d-1, wi], wi ] & ( ( ( 1 << d ) - 1 ) << ( (d :
213
214                         LF[8*fi + k, j] = v
215
216     return
217
218
219
220
221
222     flocal_hist_shape = ( 1<<max_depth, 2, 32 )
223     @cuda.jit
224     def _unweighted_featureless_histogram(Y, LE, H0, stride):
225
226         tree_set = cuda.blockIdx.x
227         ti        = cuda.blockIdx.y
228
229         wi = cuda.threadIdx.x
230
231         hist = cuda.shared.array( shape=flocal_hist_shape, dtype=np.int32 )
232
233
234         for i in range( 1<<max_depth ):
235             hist[i, 0, wi] = 0
236             hist[i, 1, wi] = 0
237
238
239
240         for j in range( stride ):

```

```

241     jj = 32*stride*ti + 32*j + wi
242
243     if jj < Y.shape[0]:
244
245         lk = LE[tree_set, jj]
246         y = Y [          jj]
247
248
249
250         for d in range(max_depth):
251             to = ( 1 << d ) - 1
252             tk = lk & to
253             lk = lk >> d
254
255             hist[ to + tk, 0, wi ] += y
256             hist[ to + tk, 1, wi ] += 1
257
258
259
260     for k in range( (1<<max_depth) - 1 ):
261
262         cuda.atomic.add( H0, (tree_set, k, 0, ), np.int64( hist[k, 0, wi] ) )
263         cuda.atomic.add( H0, (tree_set, k, 1, ),          hist[k, 1, wi] )
264
265     return
266
267
268
269
270     shared_hist_shape = ( max( (1<<max_depth) - (1<<3),1) , 2, 32 )
271
272
273     @cuda.jit
274     def _unweighted_histogram(XS, Y, L, H, warps_per_block, stride):
275
276         feat_set = cuda.blockIdx.x
277         ti       = cuda.blockIdx.y
278
279
280         wi       = cuda.threadIdx.x & 31
281         bi       = cuda.threadIdx.x >> 5
282
283         ti = warps_per_block*ti + bi
284
285
286
287         hf0 = np.int32(0)
288         hw0 = np.int32(0)
289
290         hf10 = np.int32(0)
291         hf11 = np.int32(0)
292         hw10 = np.int32(0)
293         hw11 = np.int32(0)
294
295
296         hf20 = np.int32(0)
297         hf21 = np.int32(0)
298         hf22 = np.int32(0)

```

```

299     hf23 = np.int32(0)
300     hw20 = np.int32(0)
301     hw21 = np.int32(0)
302     hw22 = np.int32(0)
303     hw23 = np.int32(0)
304
305
306     shared_hist = cuda.shared.array(shape=shared_hist_shape, dtype=np.int32)
307
308
309     for i in range( (1<max_depth) - (1<3) ):
310         shared_hist[i, 0, wi] = 0
311         shared_hist[i, 1, wi] = 0
312
313
314
315     cuda.syncthreads()
316
317
318
319     for j in range( stride ):
320
321         if 32*(stride*ti + j) < XS.shape[1]:
322
323             jj = 32*stride*ti + 32*j + wi
324
325             if jj < Y.shape[0]:
326                 lvd = L[feat_set, jj]
327                 y = np.int32(Y[          jj])
328
329                 xfd = XS[feat_set, jj]
330
331
332                 for k in range(32):
333
334                     jj = 32*stride*ti + 32*j + k
335
336
337                     if jj < Y.shape[0]:
338
339                         v = np.int32( xfd & 1 )
340                         xfd = xfd >> 1
341
342
343                         yk = cuda.shfl_sync( -1, y, k )
344                         lk = cuda.shfl_sync( -1, lvd, k )
345
346
347                         #d0
348                         hf0 += v*yk
349                         hw0 += v
350
351                         #d1
352                         tk = lk & 1
353
354                         if tk==0:
355                             hf10 += v*yk
356                             hw10 += v

```



```

357         else:
358             hf11 += v*yk
359             hw11 += v
360
361         lk = lk >> 1
362
363         #d2
364         tk = lk & 3
365
366         if tk==0:
367             hf20 += v*yk
368             hw20 += v
369         elif tk==1:
370             hf21 += v*yk
371             hw21 += v
372         elif tk==2:
373             hf22 += v*yk
374             hw22 += v
375         else:
376             hf23 += v*yk
377             hw23 += v
378
379         lk = lk >> 2
380
381
382
383         for d in range( 3, max_depth ):
384             to = ( 1 << d ) - 1
385             tk = lk & to
386             lk = lk >> d
387
388             cuda.atomic.add( shared_hist, (to + tk - 7, 0, wi ), v*yk )
389             cuda.atomic.add( shared_hist, (to + tk - 7, 1, wi ), v )
390
391
392
393
394
395         cuda.atomic.add( H, (feat_set, 0, 0, wi), hf0)
396
397         cuda.atomic.add( H, (feat_set, 0, 1, wi), hw0)
398
399
400         cuda.atomic.add( H, (feat_set, 1, 0, wi), hf10)
401         cuda.atomic.add( H, (feat_set, 2, 0, wi), hf11)
402
403         cuda.atomic.add( H, (feat_set, 1, 1, wi), hw10)
404         cuda.atomic.add( H, (feat_set, 2, 1, wi), hw11)
405
406
407         cuda.atomic.add( H, (feat_set, 3, 0, wi), hf20)
408         cuda.atomic.add( H, (feat_set, 4, 0, wi), hf21)
409         cuda.atomic.add( H, (feat_set, 5, 0, wi), hf22)
410         cuda.atomic.add( H, (feat_set, 6, 0, wi), hf23)
411
412         cuda.atomic.add( H, (feat_set, 3, 1, wi), hw20)
413         cuda.atomic.add( H, (feat_set, 4, 1, wi), hw21)
414         cuda.atomic.add( H, (feat_set, 5, 1, wi), hw22)

```

```

415     cuda.atomic.add( H, (feat_set, 6, 1, wi), hw23)
416
417
418
419
420     cuda.syncthreads()
421
422     n = (1<<max_depth) - 8
423     w = ( n + warps_per_block - 1 ) // warps_per_block
424
425     for k in range( w ):
426
427         i = w*bi + k + 7
428
429         if i < H.shape[1]:
430
431             cuda.atomic.add( H, (feat_set, i, 0, wi), np.int64( shared_hist[ i - 7, 0,
432             cuda.atomic.add( H, (feat_set, i, 1, wi),                shared_hist[ i - 7, 1,
433
434     return
435
436
437
438
439 @cuda.jit
440 def _cut_cuda( F, FST, H, H0, V, I, tree_set, L2, lr):
441
442     leaf = cuda.blockIdx.x
443     wi = cuda.threadIdx.x
444
445
446     depth = 31 - cuda.clz( np.uint32( leaf + 1 ) )
447
448     mxs = cuda.local.array( shape=( 20 ),      dtype=np.int32)
449     vrs = cuda.local.array( shape=( 20 ),      dtype=np.int32)
450     vls = cuda.local.array( shape=( 20 ),      dtype=np.int32)
451     fs = cuda.local.array( shape=( 20 ),      dtype=np.uint16)
452
453
454
455     g0ls = cuda.local.array( shape=( 20 ),      dtype=np.float32)
456     n0ls = cuda.local.array( shape=( 20 ),      dtype=np.float32)
457
458
459
460
461
462     for k in range(H0.shape[0]):
463         mxs[k] = -1e8
464
465         g0ls[k] = np.float32( H0[k, leaf, 0] )
466         n0ls[k] = np.float32( H0[k, leaf, 1] )
467
468
469     L2 = np.float32(L2 ) * 2.** ( 5 - depth)
470
471
472

```

```

473     for k in range(H.shape[0]):
474
475         tree_fold = FST[tree_set, k, depth]
476
477
478         G0 = np.float32( H[k, leaf, 0, wi] )
479         N0 = np.float32( H[k, leaf, 1, wi] )
480
481         G01 = g0ls[tree_fold]
482         N01 = n0ls[tree_fold]
483
484
485         G1 = G01 - G0
486         N1 = N01 - N0
487
488         V0 = G0 / ( N0 + L2 )
489         V1 = G1 / ( N1 + L2 )
490
491         S0 = G0 * V0
492         S1 = G1 * V1
493
494
495         S = np.float32( (S0 + S1) ).view(np.int32)
496
497         if ( mxs[tree_fold] < S ):
498
499             mxs[tree_fold] = S
500             vls[tree_fold] = int( lr * V0 * ( 1 << ( 31 - qgrad_bits ) ) * 2**( - np.
501             vrs[tree_fold] = int( lr * V1 * ( 1 << ( 31 - qgrad_bits ) ) * 2**( - np.
502             fs [tree_fold] = k
503
504
505
506
507     for tree_fold in range(V.shape[1]):
508
509         mx = mxs[tree_fold]
510
511         mxw = mx
512         cuda.syncwarp(-1)
513
514         for p in range(5):
515             v = cuda.shfl_xor_sync(-1, mxw, 1<<p )
516
517             mxw = max(v, mxw)
518
519         msk = cuda.ballot_sync(-1, mx == mxw )
520
521         is_max = ( mx == mxw ) and ( ( 1 << wi ) > ( msk >> 1 ) )
522
523
524
525         if is_max:
526
527             V[tree_set, tree_fold, 2*leaf ] = vls[tree_fold]
528             V[tree_set, tree_fold, 2*leaf + 1 ] = vrs[tree_fold]
529
530

```

```

531         I [tree_set, tree_fold, leaf ] = F[tree_set, 32*fs[tree_fold] + wi ]
532
533     return
534
535
536
537 @cuda.jit
538 def _advance_and_predict( P, X, L_old, L_new, V, I, stride, tree_set,):
539
540     tree_fold = cuda.blockIdx.x
541     depth      = cuda.blockIdx.y
542     i          = cuda.blockIdx.z
543
544     wi = cuda.threadIdx.x
545
546     for j in range(stride):
547         k = 32*stride*i + 32*j + wi
548
549         if k < L_old.shape[2]:
550
551             leaf = np.uint16( L_old[tree_fold, depth-1, k] if depth > 0 else 0 )
552
553             lo = leaf + (1<<depth) - 1
554
555             li = I[tree_set, tree_fold, lo]
556
557
558             x = ( X[li, k>>5] >> ( k&31 ) ) & 1
559
560
561             leaf = 2*leaf + x
562
563             if depth < L_new.shape[1]:
564                 L_new[tree_fold, depth, k] = tree_dtype( leaf )
565
566
567             cuda.atomic.add( P, k, V[tree_set, tree_fold, 2*lo + 1 + x ] )
568
569     return
570
571
572
573
574
575 self._et_sample                = _et_sample_1b
576 self._prep_vars                = _prep_vars
577 self._repack_trees_for_features = _repack_trees_for_features
578 self._unweighted_featureless_histogram = _unweighted_featureless_histogram
579 self._unweighted_histogram    = _unweighted_histogram
580 self._cut_cuda                 = _cut_cuda
581 self._advance_and_predict      = _advance_and_predict
582
583
584
585
586 self.tree_set = 0
587
588

```

```

589
590
591 def _fit(self, rounds):
592
593     for k in range(rounds):
594
595         assert(self.tree_set < self.dFST.shape[0])
596
597         self.dH = cuda.to_device( np.zeros([self.nfeatsets, 2**self.max_depth, 2, 32
598
599
600         self.dH0 = cuda.to_device( np.zeros([self.nfolds,      2**self.max_depth, 2
601
602
603
604
605         strides = 256
606
607         stride = int( np.ceil( self.dXS.shape[1] / strides / 32 ) )
608
609         self._et_sample[ ( self.nfeatsets , strides), 32]( self.dX, self.dXS, self.dF
610
611
612
613
614         strides = 512
615
616         stride = int( np.ceil( self.dY.shape[0] / strides / 32 ) )
617
618         self._prep_vars[strides, 32]( self.dL, self.dLE, self.dY, self.dP, self.dG, s
619
620
621
622         strides = 1024
623
624         stride = int( np.ceil( self.dY.shape[0] / strides / 32 ) )
625
626         self._repack_trees_for_features[( (self.dFST.shape[1]+7)//8, strides), 32]( se
627
628
629
630         warps_per_block = self.hist_warps_per_block
631         blocks_per_feat = ( (64*103 )//warps_per_block + self.nfeatsets - 1) // self.i
632
633         strides = blocks_per_feat * warps_per_block
634
635         stride = int( np.ceil( self.dXS.shape[1] / strides / 32 ) )
636
637         self._unweighted_histogram[ ( self.dXS.shape[0], blocks_per_feat), warps_per_b
638
639
640         strides = 512
641
642         stride = int( np.ceil( self.dY.shape[0] / strides / 32 ) )
643
644         self._unweighted_featureless_histogram[(self.nfolds, strides), 32](self.dG, se
645
646

```

```

647
648
649     self._cut_cuda[2**self.max_depth - 1, 32]( self.dF, self.dFST, self.dH, self.d
650
651
652
653
654     strides = 512
655
656     stride = int( np.ceil( self.dP.shape[0] / strides / 32 ) )
657
658     self._advance_and_predict[(self.nfolds, min(self.tree_set+1, self.max_depth), :
659
660
661     self.dL, self.dLn = self.dLn, self.dL
662
663
664     if self.dXv is not None:
665
666         stride = int( np.ceil( self.dPv.shape[0] / strides / 32 ) )
667
668         self._advance_and_predict[(self.nfolds, min(self.tree_set+1, self.max_depth)
669
670
671         self.dLv, self.dLvn = self.dLvn, self.dLv
672
673
674     for callback in self.callbacks:
675         callback(self)
676
677
678     self.tree_set = self.tree_set + 1
679
680
681
682 def fit(self, X, Y, Xv=None, Yv=None, F=None, FST=None, C=None, rounds=None, callb
683
684
685     rounds = ( F.shape[0] if rounds is None else rounds )
686
687
688
689     self.callbacks = callbacks
690
691     self.dX = cuda.to_device( X )
692
693     self.dXS = cuda.to_device(np.zeros([ self.nfeatsets, 32*X.shape[1] ], np.uint32
694
695
696     self.dH0 = cuda.to_device( np.zeros([ self.nfolds, 2**self.max_depth, 2 ], dtype
697
698
699
700     self.dF = cuda.to_device(F)
701
702     self.dL = cuda.to_device( np.zeros([self.nfolds, self.max_depth-1, Y.shape[0]],
703     self.dLE = cuda.to_device( np.zeros([self.nfolds,
704

```

```

705 self.dY = cuda.to_device( (Y*(1<30)).astype(np.int32) )
706 self.dP = cuda.to_device( np.zeros(Y.shape, dtype=np.int32) )
707 self.dG = cuda.to_device( np.zeros(Y.shape, dtype=self.grad_dtype) )
708
709 self.dH = cuda.to_device( np.zeros([self.nfeatsets, 2**self.max_depth, 2, 32]
710 self.dLF = cuda.to_device( np.zeros([self.nfeatsets, Y.shape[0]], dtype=self.p
711 self.dFST = cuda.to_device(FST.astype(np.uint8))
712
713
714
715 self.dV = cuda.to_device(np.zeros([rounds, self.nfolds, 2*(2**self.max_depth - 1
716 self.dI = cuda.to_device(np.zeros([rounds, self.nfolds, (2**self.max_depth - 1
717
718 self.dLn = cuda.to_device( np.zeros(self.dL.shape, dtype=self.tree_dtype) )
719
720
721 if Xv is not None:
722
723     self.dXv = cuda.to_device( Xv )
724
725     self.dPv = cuda.to_device( np.zeros(Yv.shape, dtype=np.int32) )
726     self.Yv = Yv
727
728     self.dLv = cuda.to_device( np.zeros([self.nfolds, self.max_depth-1, Yv.shape[0
729     self.dLvN = cuda.to_device( np.zeros([self.nfolds, self.max_depth-1, Yv.shape[0
730 else:
731     self.dXv = None
732
733
734 self._fit(rounds)
735
736
737
738
739 def predict(self, X, shape=None):
740
741
742     if shape is None:
743         shape = 32*X.shape[1]
744
745     dX = cuda.to_device( X )
746     dL = cuda.to_device( np.zeros([self.nfolds, self.max_depth-1, shape], dtype=self
747     dLn = cuda.to_device( np.zeros([self.nfolds, self.max_depth-1, shape], dtype=self
748     dP = cuda.to_device( np.zeros([shape], dtype=np.int32) )
749
750
751     for k in range( self.tree_set ):
752
753         strides = 256
754
755         stride = int( np.ceil( dP.shape[0] / strides / 32 ) )
756
757         self._advance_and_predict([self.nfolds, min(k+1, self.max_depth), strides), 32
758
759         dL, dLn = dLn, dL
760
761     p = dP.copy_to_host()
762

```

```

763     del dX, dL, dLn, dP
764
765     return p
766
767
768
769
770 def save(self, path):
771
772     V = self.dV.copy_to_host()
773     I = self.dI.copy_to_host()
774
775     V = V[:self.tree_set]
776     I = I[:self.tree_set]
777
778     np.savez(path, V=V, I=I)
779
780
781 def load(self, path):
782
783     data = np.load(path)

```

✓ Logger

```

1 class LoggingCallback(object):
2
3     def __init__(self, frequency=500):
4         self.frequency = frequency
5         self.val_corrs = []
6         self.trn_corrs = []
7
8     def __call__(self, booster):
9
10
11         round = booster.tree_set + 1
12
13         if booster.tree_set == 0:
14             self.t0 = time.time()
15             self.t = self.t0
16
17         if round % self.frequency == 0:
18
19             ct = np.corrcoef( booster.dP.copy_to_host(), booster.dY.copy_to_host() )[0,1]
20
21             print( 'Round: {} --- Trees {}'.format( round, booster.nfolds*round ) )
22             print()
23             print( 'Train Corr: {:.4f}'.format(ct) )
24
25             self.trn_corrs.append( ct )
26
27             if booster.dXv is not None:
28                 pv = booster.dPv.copy_to_host()
29                 yv = booster.Yv
30
31                 cv = np.corrcoef( pv, yv )[0,1]
32
33                 print( 'Valid Corr: {:.4f}'.format(cv) )

```



```

34
35
36     self.val_corrs.append( cv )
37
38     elapsed = time.time() - self.t0
39     dt      = time.time() - self.t
40
41     print()
42     print( 'Time: {:.2f}s  --  Step: {:.2f}s  --  TPS: {:.1f}'.format( elapsed, dt,
43     print()
44     print()
45     print()
46     self.t = time.time()

```

1 Start coding or generate with AI.

✓ Bitpacked Encoding

```

1 @cuda.jit
2 def _encode_cuts( X, XB, stride, ):
3
4     f = cuda.blockIdx.x
5     bi = cuda.blockIdx.y
6     wi = cuda.threadIdx.x
7
8     sm = cuda.shared.array(shape=( 32, 32 ), dtype=np.uint32)
9
10
11     for i in range(stride):
12
13         i_in = 32*32*stride*bi + 32*32*i + wi
14         i_out = 32*stride*bi + 32*i + wi
15
16         v0 = np.uint32(0)
17         v1 = np.uint32(0)
18         v2 = np.uint32(0)
19         v3 = np.uint32(0)
20
21
22         for k in range(32):
23
24             sm[wi, (k+wi)%32] = X[f, 32*k + i_in ] if 32*k + i_in < X.shape[1] else 0
25
26
27         for k in range(32):
28
29             v = sm[k, (k+wi)%32]
30
31             v0 |= (1 if v>0 else 0 ) << k
32             v1 |= (1 if v>1 else 0 ) << k
33             v2 |= (1 if v>2 else 0 ) << k
34             v3 |= (1 if v>3 else 0 ) << k
35
36
37         if i_out < XB.shape[1]:
38             XB[4*f + 0, i_out] = v0
39             XB[4*f + 1, i_out] = v1

```

```

40     XB[4*f + 2, i_out] = v2
41     XB[4*f + 3, i_out] = v3
42
43
44 def encode_cuts(X):
45
46     X = (X.T).copy()
47
48     strides = 64
49
50     dX = cuda.to_device(X)
51     dXB = cuda.to_device( np.zeros([4*dX.shape[0], (dX.shape[1]+31)//32 ], dtype=np.uint8) )
52
53
54     stride = int( np.ceil( dX.shape[1] / 32**2 / strides ) )
55
56     _encode_cuts[(X.shape[0], strides), 32 ]( dX, dXB, stride)
57
58     X = dXB.copy_to_host()
59
60     del dX, dXB
61     gc.collect()
62
63
64     return X

```

✓ Model Parameters and Feature Pre-Selection

```

1 lr = 0.07
2 trees_per_layer = 8
3 nsets = 10000
4 nfeatsets = 16*2
5 max_depth = 7
6 L2 = 100_000
7
8
9 # feature schedule for booster
10 F = np.array( [ np.hstack([ np.tile(np.random.choice(4*Xt.shape[1], 32, replace=True)
11                             for k in range(nsets) ) ] )
12 F = F.astype(np.uint16)
13
14
15 # 32 feature subset to tree batch map by depth
16 FST = np.tile( np.tile( np.arange(trees_per_layer), [ (nfeatsets + trees_per_layer - 1)
17
18 [[np.random.shuffle(FST[k,kk]) for kk in range(max_depth)] for k in range(nsets)];
19 FST = FST.transpose(0,2,1).copy()
20
21 feats_per_layer = 32 * nfeatsets // trees_per_layer
22 feats_per_layer

```

↻ 128

✓ Training

```

1
2 Xte = encode_cuts(Xt)

```

```

3 Xve = encode_cuts(Xv)
4
5
6 logger = LoggingCallback()
7
8
9 booster = ExtraFastBooster( trees_per_layer=trees_per_layer, max_depth=max_depth, nfe:
10
11 booster.fit(Xte, Yt, Xve, Yv, F, FST, callbacks=[logger])
12
13 booster.save('saved_booster.npz')
14
15
16 pv = booster.predict(Xve, Yv.shape[0])
17
18
19 np.corrcoef(pv, Yv)

```

 /usr/local/lib/python3.10/dist-packages/numba/cuda/dispatcher.py:536: NumbaPerformanceWarning: Grid size 127 will likely result in poor performance. Consider using a power of 2 grid size.
warn(NumbaPerformanceWarning(msg))
Round: 500 --- Trees 4000

Train Corr: 0.1023
Valid Corr: 0.0246

Time: 15.16s -- Step: 15.16s -- TPS: 263.8

Round: 1000 --- Trees 8000

Train Corr: 0.1308
Valid Corr: 0.0280

Time: 30.43s -- Step: 15.27s -- TPS: 262.0

Round: 1500 --- Trees 12000

Train Corr: 0.1531
Valid Corr: 0.0300

Time: 45.61s -- Step: 15.18s -- TPS: 263.5

Round: 2000 --- Trees 16000

Train Corr: 0.1721
Valid Corr: 0.0314

Time: 60.80s -- Step: 15.19s -- TPS: 263.4

Round: 2500 --- Trees 20000

Train Corr: 0.1890
Valid Corr: 0.0324

Time: 75.99s -- Step: 15.19s -- TPS: 263.3

Round: 3000 --- Trees 24000

Train Corr: 0.2043
Valid Corr: 0.0332

Time: 91.19s -- Step: 15.20s -- TPS: 263.2

Round: 3500 --- Trees 28000

Offline Model

```

1  @jit(nopython=True)
2  def pack_cuts_32(X, n):
3
4      Xo = np.zeros( ( n*X.shape[0], ( X.shape[1]+ 31 )>>5 ) , np.uint32)
5
6      for f in range( X.shape[0] ):
7          for k in range( X.shape[1] ):
8
9              x = X[f, k]
10
11              for s in range(n):
12
13                  Xo[n*f + s, k>>5] |= ( 1 << ( k&31 ) ) * ( x > s )
14
15
16      return Xo
17
18
19
20
21 @numba.jit(nopython=True)
22 def _advance_and_predict_offline(X, P, I, V, L_old, L_new, tree_set, max_depth):
23
24     for depth in range(max_depth):
25         for tree_fold in range(I.shape[1]):
26             for k in range(L_new.shape[-1]):
27
28                 leaf = ( L_old[tree_fold, depth-1, k] if depth > 0 else 0 )
29
30                 lo = leaf + (1<<depth) - 1
31
32                 li = I[tree_set, tree_fold, lo]
33
34                 x = ( X[li, k>>5] >> ( k&31 ) ) & 1
35
36                 leaf = 2*leaf + x
37
38                 if depth < L_new.shape[1]:
39                     L_new[tree_fold, depth, k] = leaf
40
41
42                 P[k] += V[tree_set, tree_fold, 2*lo + 1 + x ]
43
44     return
45
46
47 def predict_offline(X, I, V, max_depth, n):
48     P = np.zeros((n,), np.int32)
49     L_old = np.zeros(( I.shape[1], max_depth-1, n), np.uint16)
50     L_new = np.zeros(( I.shape[1], max_depth-1, n), np.uint16)
51
52
53     for k in range(I.shape[0]):
54
55         _advance_and_predict_offline(X, P, I, V, L_old, L_new, k, min(k+1, max_depth))
56

```

```

57     L_new, L_old = L_old, L_new
58
59     return P
60
61
62 class OfflineExtraFastBooster(object):
63
64
65     def __init__(self, path):
66         data = np.load(path)
67
68         self.V = data['V']
69         self.I = data['I']
70         self.max_depth = int.bit_count(self.I.shape[-1])
71
72
73     def predict(self, X, n=None):
74
75         n = 32*X.shape[1] if n is None else n
76
77         return predict_offline(X, self.I, self.V, self.max_depth, n)
78
79
80

```

✓ Offline Prediction

Verifying offline preds for one era:

```

1  off = OfflineExtraFastBooster('saved_booster.npz')
2
3  b = Ev==np.min(Ev)
4
5  Xve = Xv[b].T
6
7  Xve = pack_cuts_32(Xve, 4)
8
9  pvo = off.predict(Xve, b.sum() )
10
11 np.corrcoef(pvo, pv[b])

```

```

↔ array([[1., 1.],
        [1., 1.]])

```

✓ Upload Diagnostics

```

1  pv = pv-np.min(pv)
2
3  pv = pv/np.max(pv)
4
5  pv = pv*.98+.01
6
7
8  df_sub['prediction'] = pv[:df_sub.shape[0]]
9
10
11 df_sub.to_csv('predictions.csv')

```

```
12 submission_id = napi.upload_diagnostics("predictions.csv", model_id=napi.get_models())
```