Imports

```
!pip install numerapi

import matplotlib
import numpy as np
import pandas as pd
from numerapi import NumerAPI
import random
import sklearn
import time
import lightgbm
import matplotlib.pyplot as plt
import gc

from numba import cuda, jit
import numba



%matplotlib inline

```

```
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)
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

```
your_public_id = ""
your_secret_key = ""
your_model_slot_name = ""



napi = NumerAPI(your_public_id, your_secret_key)

current_round = napi.get_current_round()

napi.download_dataset('v5.0/features.json')
napi.download_dataset('v5.0/train.parquet')
napi.download_dataset('v5.0/validation.parquet')
```

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

Data

```
import json

ff =  json.load(open('v5.0/features.json', 'rb'))


fs = ff['feature_sets']['all']


df_train = pd.read_parquet('v5.0/train.parquet')

df_train['era'] = df_train['era'].astype(int)


df_train = df_train[df_train.era < df_train.era.max() - 4]

Xt = df_train[fs].fillna(2).astype(np.uint8).values
Yt = df_train['target'].fillna(0).values
Yt = 2*Yt - 1
Et = df_train['era'].values

del df_train
gc.collect()

df_valid = pd.read_parquet('v5.0/validation.parquet')

Xv = df_valid[fs].fillna(2).astype(np.uint8).values
Yv = df_valid['target'].fillna(0).values
Yv = 2*Yv - 1
Ev = df_valid['era'].values



df_sub = df_valid[[]]

del df_valid
gc.collect()


```

```
0
```

Cuda Model Definition

```


class ExtraFastBooster(object):

  def __init__(self, trees_per_layer=8, max_depth=6, nfeatsets=8, lr=.01, L2=10_000,

    self.L2 = L2
    self.lr = lr
```

```
    self.max_depth = max_depth
    self.nfeatsets = nfeatsets
    self.nfolds = trees_per_layer


    qgrad_mbits = qgrad_bits - 1


    if   max_depth > 8:
      packed_tree_dtype = np.uint64
    elif max_depth > 6:
      packed_tree_dtype = np.uint32
    else:
      packed_tree_dtype = np.uint16

    if max_depth > 8 :
      tree_dtype = np.uint16
    else:
      tree_dtype = np.uint8



    grad_dtype        = ( np.int16 if qgrad_mbits>7 else np.int8   )
    grad_mbits        = (       15 if qgrad_mbits>7 else       7   )
    ymax_lg2          = 30



    residual_sm_a100 = 15 - 2 - 6 - max_depth


    assert( residual_sm_a100 >= 0 )



    hist_warps_per_block_lg2 = max( 4 - max( residual_sm_a100, 0), 0)




    self.grad_dtype = grad_dtype
    self.tree_dtype = tree_dtype
    self.packed_tree_dtype = packed_tree_dtype
    self.hist_warps_per_block = 1 << hist_warps_per_block_lg2



    @cuda.jit
    def _et_sample_1b(  X, XS, F, tree_set, stride, ):

      f0 = cuda.blockIdx.x
      bi = cuda.blockIdx.y
      wi = cuda.threadIdx.x



      fs = cuda.shared.array(shape=32, dtype=np.uint32)
      sm = cuda.shared.array(shape=( 32, 32 ), dtype=np.uint32)
```

```
    fs[wi] = F[tree_set, 32*f0 + wi]

    cuda.syncwarp()

    for i in range(stride):

      i_in = 32*stride*bi + 32*i + wi

      if 32*stride*bi + 32*i < X.shape[1]:



        for k in range(4):
          ff0 = fs[8*k + 0]
          ff1 = fs[8*k + 1]
          ff2 = fs[8*k + 2]
          ff3 = fs[8*k + 3]
          ff4 = fs[8*k + 4]
          ff5 = fs[8*k + 5]
          ff6 = fs[8*k + 6]
          ff7 = fs[8*k + 7]

          kk0 = (wi + 8*k + 0)%32
          kk1 = (wi + 8*k + 1)%32
          kk2 = (wi + 8*k + 2)%32
          kk3 = (wi + 8*k + 3)%32
          kk4 = (wi + 8*k + 4)%32
          kk5 = (wi + 8*k + 5)%32
          kk6 = (wi + 8*k + 6)%32
          kk7 = (wi + 8*k + 7)%32

          v0 = ( X[ff0, i_in] if i_in < X.shape[1] else 0 )
          v1 = ( X[ff1, i_in] if i_in < X.shape[1] else 0 )
          v2 = ( X[ff2, i_in] if i_in < X.shape[1] else 0 )
          v3 = ( X[ff3, i_in] if i_in < X.shape[1] else 0 )
          v4 = ( X[ff4, i_in] if i_in < X.shape[1] else 0 )
          v5 = ( X[ff5, i_in] if i_in < X.shape[1] else 0 )
          v6 = ( X[ff6, i_in] if i_in < X.shape[1] else 0 )
          v7 = ( X[ff7, i_in] if i_in < X.shape[1] else 0 )

          sm[8*k+0, kk0] = v0
          sm[8*k+1, kk1] = v1
          sm[8*k+2, kk2] = v2
          sm[8*k+3, kk3] = v3
          sm[8*k+4, kk4] = v4
          sm[8*k+5, kk5] = v5
          sm[8*k+6, kk6] = v6
          sm[8*k+7, kk7] = v7



        cuda.syncwarp()

```

```
          for k in range(32):
            i_out = 32*32*stride*bi + 32*32*i + 32*k + wi

            kk = (k + wi)%32

            XS[f0, i_out] =  sm[wi, kk]









    @cuda.jit
    def _prep_vars(  L, LE, Y, P, G, stride ):

      ti = cuda.blockIdx.x
      wi = cuda.threadIdx.x


      for i in range( stride ):

        j = 32*stride*ti + 32*i + wi


        if j < Y.shape[0]:


          for k in range(LE.shape[0]):

            v = packed_tree_dtype(0)
            for d in range(1, max_depth):
              v |= L[ k, d-1, j]  << ( ( d*(d-1) )//2 )

            LE[k, j] = v



          g = ( np.int64(Y[j]) - np.int64(P[j]) ) >> ( 31 - qgrad_mbits )

          g = min( max(g, - ( 1 << grad_mbits ) + 1 ), ( 1 << grad_mbits) - 1 )

          G[j] = grad_dtype( g )

      return




    fstmem = ( 8, max_depth-1, 32 )


    @cuda.jit
    def _repack_trees_for_features( FST,  LE, LF, tree_set, stride ):

      fi = cuda.blockIdx.x
      ti = cuda.blockIdx.y
```

```
    wi = cuda.threadIdx.x

    fst = cuda.shared.array( shape=fstmem,     dtype=np.uint32)

    trees = cuda.shared.array( shape=(20, 32), dtype= packed_tree_dtype )


    for k in range(8):
      if 8*fi + k < FST.shape[1]:
        for d in range(1, FST.shape[2]):
          fst[k, d-1, wi] = FST[tree_set, 8*fi + k, d]



    for i in range( stride ):

      j = 32*stride*ti + 32*i + wi

      if j < LF.shape[1]:


        for k in range(LE.shape[0]):
          trees[k,wi] = LE[k, j]

        for k in range(8):

          if 8*fi + k < LF.shape[0]:
            v = 0
            for d in range(1, FST.shape[2]):
              v |= trees[ fst[k, d-1, wi], wi ] & (  (  ( 1 << d ) - 1 ) << (  (d 

            LF[8*fi + k, j] = v

    return




flocal_hist_shape = ( 1<<max_depth, 2, 32 )
@cuda.jit
def _unweighted_featureless_histogram(Y, LE, H0, stride):

    tree_set = cuda.blockIdx.x
    ti       = cuda.blockIdx.y

    wi = cuda.threadIdx.x

    hist  = cuda.shared.array( shape=flocal_hist_shape,  dtype=np.int32 )


    for i in range( 1<<max_depth ):
      hist[i, 0, wi] = 0
      hist[i, 1, wi] = 0



    for j in range( stride ):
```

```
        jj = 32*stride*ti + 32*j + wi

        if jj < Y.shape[0]:


          lk = LE[tree_set,  jj]
          y  = Y [           jj]


          for d in range(max_depth):
            to = ( 1 << d ) - 1
            tk = lk & to
            lk = lk >> d

            hist[ to + tk,  0, wi ] += y
            hist[ to + tk,  1, wi ] += 1



      for k in range( (1<<max_depth) - 1 ):

        cuda.atomic.add( H0, (tree_set, k, 0, ), np.int64( hist[k, 0, wi] ) )
        cuda.atomic.add( H0, (tree_set, k, 1, ),           hist[k, 1, wi] )

      return




    shared_hist_shape = (  max( (1<<max_depth) - (1<<3),1) , 2, 32 )


    @cuda.jit
    def _unweighted_histogram(XS, Y, L, H, warps_per_block, stride):

      feat_set  = cuda.blockIdx.x
      ti        = cuda.blockIdx.y


      wi         = cuda.threadIdx.x & 31
      bi         = cuda.threadIdx.x >> 5

      ti = warps_per_block*ti + bi



      hf0 = np.int32(0)
      hw0 = np.int32(0)

      hf10 = np.int32(0)
      hf11 = np.int32(0)
      hw10 = np.int32(0)
      hw11 = np.int32(0)


      hf20 = np.int32(0)
      hf21 = np.int32(0)
      hf22 = np.int32(0)
```

```
    hf23 = np.int32(0)
    hw20 = np.int32(0)
    hw21 = np.int32(0)
    hw22 = np.int32(0)
    hw23 = np.int32(0)


    shared_hist = cuda.shared.array(shape=shared_hist_shape, dtype=np.int32)


    for i in range( (1<<max_depth) - (1<<3) ):
      shared_hist[i, 0, wi] = 0
      shared_hist[i, 1, wi] = 0



    cuda.syncthreads()



    for j in range( stride ):

        if 32*(stride*ti + j) < XS.shape[1]:

          jj = 32*stride*ti + 32*j + wi

          if jj < Y.shape[0]:
            lvd = L[feat_set,  jj]
            y  = np.int32(Y[        jj])

          xfd = XS[feat_set, jj]


          for k in range(32):

            jj = 32*stride*ti + 32*j + k


            if jj < Y.shape[0]:

              v   = np.int32( xfd  & 1 )
              xfd = xfd >> 1


              yk = cuda.shfl_sync( -1,   y, k )
              lk = cuda.shfl_sync( -1, lvd, k )


              #d0
              hf0 += v*yk
              hw0 += v

              #d1
              tk = lk & 1

              if tk==0:
                hf10 += v*yk
                hw10 += v
```

```
                else:
                  hf11 += v*yk
                  hw11 += v

                lk = lk >> 1

                #d2
                tk = lk  & 3

                if   tk==0:
                  hf20 += v*yk
                  hw20 += v
                elif tk==1:
                  hf21 += v*yk
                  hw21 += v
                elif tk==2:
                  hf22 += v*yk
                  hw22 += v
                else:
                  hf23 += v*yk
                  hw23 += v

                lk = lk >> 2



                for d in range( 3, max_depth ):
                  to = ( 1 << d ) - 1
                  tk = lk & to
                  lk = lk >> d

                  cuda.atomic.add( shared_hist, (to + tk - 7,  0,  wi ), v*yk )
                  cuda.atomic.add( shared_hist, (to + tk - 7,  1,  wi ), v    )




      cuda.atomic.add( H, (feat_set, 0, 0, wi), hf0)

      cuda.atomic.add( H, (feat_set, 0, 1, wi), hw0)


      cuda.atomic.add( H, (feat_set, 1, 0, wi), hf10)
      cuda.atomic.add( H, (feat_set, 2, 0, wi), hf11)

      cuda.atomic.add( H, (feat_set, 1, 1, wi), hw10)
      cuda.atomic.add( H, (feat_set, 2, 1, wi), hw11)


      cuda.atomic.add( H, (feat_set, 3, 0, wi), hf20)
      cuda.atomic.add( H, (feat_set, 4, 0, wi), hf21)
      cuda.atomic.add( H, (feat_set, 5, 0, wi), hf22)
      cuda.atomic.add( H, (feat_set, 6, 0, wi), hf23)

      cuda.atomic.add( H, (feat_set, 3, 1, wi), hw20)
      cuda.atomic.add( H, (feat_set, 4, 1, wi), hw21)
      cuda.atomic.add( H, (feat_set, 5, 1, wi), hw22)
```

```
    cuda.atomic.add( H, (feat_set, 6, 1, wi), hw23)




    cuda.syncthreads()

    n = (1<<max_depth) - 8
    w = ( n + warps_per_block - 1 ) // warps_per_block

    for k in range( w ):

      i = w*bi + k + 7

      if i < H.shape[1]:

        cuda.atomic.add( H, (feat_set, i, 0, wi), np.int64( shared_hist[ i - 7, 0,
        cuda.atomic.add( H, (feat_set, i, 1, wi),           shared_hist[ i - 7, 1,

    return




@cuda.jit
def _cut_cuda( F, FST, H, H0, V, I, tree_set, L2, lr):

  leaf = cuda.blockIdx.x
  wi  = cuda.threadIdx.x


  depth = 31 - cuda.clz(  np.uint32( leaf + 1 ) )

  mxs = cuda.local.array( shape=( 20 ),     dtype=np.int32)
  vrs = cuda.local.array( shape=( 20 ),     dtype=np.int32)
  vls = cuda.local.array( shape=( 20 ),     dtype=np.int32)
  fs  = cuda.local.array( shape=( 20 ),     dtype=np.uint16)



  g01s  = cuda.local.array( shape=( 20 ),     dtype=np.float32)
  n01s  = cuda.local.array( shape=( 20 ),     dtype=np.float32)




  for k in range(H0.shape[0]):
    mxs[k] = -1e8

    g01s[k] =  np.float32( H0[k, leaf, 0] )
    n01s[k] =  np.float32( H0[k, leaf, 1] )


  L2 = np.float32(L2 ) * 2.**(  5 - depth)
```

```
    for k in range(H.shape[0]):

      tree_fold = FST[tree_set,  k, depth]


      G0 = np.float32( H[k, leaf, 0, wi] )
      N0 = np.float32( H[k, leaf, 1, wi] )

      G01 = g01s[tree_fold]
      N01 = n01s[tree_fold]


      G1  = G01 - G0
      N1  = N01 - N0

      V0 = G0 / ( N0 + L2 )
      V1 = G1 / ( N1 + L2 )

      S0  = G0 * V0
      S1  = G1 * V1


      S = np.float32( (S0 + S1)   ).view(np.int32)

      if ( mxs[tree_fold] < S ):

        mxs[tree_fold] = S
        vls[tree_fold] = int(  lr * V0 * ( 1 << ( 31 - qgrad_bits ) ) * 2**( - np.
        vrs[tree_fold] = int(  lr * V1 * ( 1 << ( 31 - qgrad_bits ) ) * 2**( - np.
        fs [tree_fold] = k



    for tree_fold in range(V.shape[1]):

      mx = mxs[tree_fold]

      mxw = mx
      cuda.syncwarp(-1)

      for p in range(5):
        v = cuda.shfl_xor_sync(-1, mxw, 1<<p )

        mxw = max(v, mxw)

      msk = cuda.ballot_sync(-1, mx == mxw )

      is_max = ( mx == mxw ) and ( ( 1 << wi ) > ( msk >> 1 ) )



      if is_max:

        V[tree_set, tree_fold, 2*leaf     ] = vls[tree_fold]
        V[tree_set, tree_fold, 2*leaf + 1 ] = vrs[tree_fold]


```

```
          I [tree_set, tree_fold, leaf ] = F[tree_set, 32*fs[tree_fold] + wi ]

      return



    @cuda.jit
    def _advance_and_predict( P, X, L_old, L_new, V, I, stride, tree_set,):

      tree_fold = cuda.blockIdx.x
      depth     = cuda.blockIdx.y
      i         = cuda.blockIdx.z

      wi = cuda.threadIdx.x

      for j in range(stride):
        k = 32*stride*i + 32*j + wi

        if k < L_old.shape[2]:

          leaf = np.uint16( L_old[tree_fold, depth-1, k] if depth > 0 else 0 )

          lo = leaf + (1<<depth) - 1

          li = I[tree_set, tree_fold, lo]


          x = ( X[li, k>>5] >> ( k&31  ) ) & 1


          leaf =  2*leaf + x

          if depth < L_new.shape[1]:
            L_new[tree_fold, depth, k] = tree_dtype( leaf )


          cuda.atomic.add( P, k, V[tree_set, tree_fold, 2*lo + 1 + x ] )

      return




    self._et_sample                       = _et_sample_1b
    self._prep_vars                       = _prep_vars
    self._repack_trees_for_features       = _repack_trees_for_features
    self._unweighted_featureless_histogram = _unweighted_featureless_histogram
    self._unweighted_histogram            =  _unweighted_histogram
    self._cut_cuda                        = _cut_cuda
    self._advance_and_predict             = _advance_and_predict



    self.tree_set = 0
```

```
589
590
591  def _fit(self, rounds):
592
593
594    for k in range(rounds):
595
596      assert(self.tree_set < self.dFST.shape[0])
597
598      self.dH  = cuda.to_device( np.zeros([self.nfeatsets, 2**self.max_depth, 2, 32
599
600
601      self.dH0 = cuda.to_device( np.zeros([self.nfolds,     2**self.max_depth, 2
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.n
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), s
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 ], dtyp
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,                   Y.shape[0]],
704
```

```
    self.dY = cuda.to_device( (Y*(1<<30)).astype(np.int32) )
    self.dP = cuda.to_device( np.zeros(Y.shape, dtype=np.int32) )
    self.dG = cuda.to_device( np.zeros(Y.shape, dtype=self.grad_dtype ) )

    self.dH   = cuda.to_device( np.zeros([self.nfeatsets, 2**self.max_depth, 2, 32 ]
    self.dLF  = cuda.to_device( np.zeros([self.nfeatsets, Y.shape[0] ], dtype=self.pa
    self.dFST = cuda.to_device(FST.astype(np.uint8))



    self.dV = cuda.to_device(np.zeros([rounds, self.nfolds, 2*(2**self.max_depth - 1
    self.dI = cuda.to_device(np.zeros([rounds, self.nfolds,   (2**self.max_depth - 1

    self.dLn = cuda.to_device( np.zeros(self.dL.shape, dtype=self.tree_dtype) )


    if Xv is not None:

      self.dXv = cuda.to_device( Xv )

      self.dPv = cuda.to_device( np.zeros(Yv.shape, dtype=np.int32) )
      self.Yv = Yv

      self.dLv  = cuda.to_device( np.zeros([self.nfolds, self.max_depth-1, Yv.shape[0
      self.dLvn = cuda.to_device( np.zeros([self.nfolds, self.max_depth-1, Yv.shape[0
    else:
      self.dXv = None


    self._fit(rounds)



  def predict(self, X, shape=None):


    if shape is None:
      shape = 32*X.shape[1]

    dX  = cuda.to_device( X )
    dL  = cuda.to_device( np.zeros([self.nfolds, self.max_depth-1, shape], dtype=self
    dLn = cuda.to_device( np.zeros([self.nfolds, self.max_depth-1, shape], dtype=self
    dP  = cuda.to_device( np.zeros([shape], dtype=np.int32 ) )


    for k in range( self.tree_set ):

      strides = 256

      stride = int( np.ceil( dP.shape[0] / strides / 32 ) )

      self._advance_and_predict[(self.nfolds, min(k+1, self.max_depth), strides), 32

      dL, dLn = dLn, dL

    p = dP.copy_to_host()

```

```
    del dX, dL, dLn, dP

    return p




  def save(self, path):

    V = self.dV.copy_to_host()
    I = self.dI.copy_to_host()

    V = V[:self.tree_set]
    I = I[:self.tree_set]

    np.savez(path, V=V, I=I)


  def load(self, path):

    data = np.load(path)
```

Logger

```
class LoggingCallback(object):

  def __init__(self, frequency=500):
    self.frequency = frequency
    self.val_corrs = []
    self.trn_corrs = []

  def __call__(self, booster):


    round = booster.tree_set + 1

    if booster.tree_set == 0:
      self.t0 = time.time()
      self.t  = self.t0

    if round % self.frequency == 0:

      ct =  np.corrcoef( booster.dP.copy_to_host(), booster.dY.copy_to_host() )[0,1]

      print( 'Round: {}  ---  Trees {}'.format( round, booster.nfolds*round ) )
      print()
      print( 'Train Corr: {:.4f}'.format(ct) )

      self.trn_corrs.append( ct )

      if booster.dXv is not None:
        pv = booster.dPv.copy_to_host()
        yv = booster.Yv

        cv = np.corrcoef( pv, yv )[0,1]

        print( 'Valid Corr: {:.4f}'.format(cv) )
```

```
        self.val_corrs.append( cv )

      elapsed = time.time() - self.t0
      dt      = time.time() - self.t

      print()
      print( 'Time: {:.2f}s  --  Step: {:.2f}s  --  TPS: {:.1f}'.format( elapsed, dt,
      print()
      print()
      print()
      self.t = time.time()
```

```
Start coding or generate with AI.
```

Bitpacked Encoding

```
@cuda.jit
def _encode_cuts(  X, XB,  stride, ):

  f  = cuda.blockIdx.x
  bi = cuda.blockIdx.y
  wi = cuda.threadIdx.x

  sm = cuda.shared.array(shape=( 32, 32 ), dtype=np.uint32)


  for i in range(stride):

    i_in  = 32*32*stride*bi + 32*32*i +  wi
    i_out =    32*stride*bi +    32*i +  wi

    v0 = np.uint32(0)
    v1 = np.uint32(0)
    v2 = np.uint32(0)
    v3 = np.uint32(0)


    for k in range(32):

      sm[wi, (k+wi)%32] = X[f, 32*k + i_in ] if 32*k + i_in < X.shape[1] else 0


    for k in range(32):

      v = sm[k, (k+wi)%32]

      v0 |=  (1 if v>0 else 0 ) << k
      v1 |=  (1 if v>1 else 0 ) << k
      v2 |=  (1 if v>2 else 0 ) << k
      v3 |=  (1 if v>3 else 0 ) << k


    if i_out < XB.shape[1]:
      XB[4*f + 0, i_out] = v0
      XB[4*f + 1, i_out] = v1
```

```
      XB[4*f + 2, i_out] = v2
      XB[4*f + 3, i_out] = v3


def encode_cuts(X):

  X = (X.T).copy()

  strides = 64

  dX  = cuda.to_device(X)
  dXB = cuda.to_device( np.zeros([4*dX.shape[0], (dX.shape[1]+31)//32  ], dtype=np.ui


  stride = int( np.ceil( dX.shape[1] / 32**2 / strides ) )

  _encode_cuts[(X.shape[0],  strides), 32 ]( dX, dXB, stride)

  X = dXB.copy_to_host()

  del dX, dXB
  gc.collect()


  return X
```

Model Parameters and Feature Pre-Selection

```
lr              = 0.07
trees_per_layer = 8
nsets     = 10000
nfeatsets = 16*2
max_depth = 7
L2 = 100_000


# feature schedule for booster
F = np.array( [  np.hstack([ np.tile(np.random.choice(4*Xt.shape[1], 32, replace=True
                   for k in range(nsets) ] )
F = F.astype(np.uint16)


# 32 feature subset to tree batch map by depth
FST = np.tile( np.tile( np.arange(trees_per_layer), [ (nfeatsets + trees_per_layer -

[[np.random.shuffle(FST[k,kk]) for kk in range(max_depth)] for k in range(nsets)];
FST = FST.transpose(0,2,1).copy()

feats_per_layer = 32 * nfeatsets // trees_per_layer
feats_per_layer
```

```
128
```

Training

```

Xte = encode_cuts(Xt)
```

```
Xve = encode_cuts(Xv)


logger = LoggingCallback()


booster = ExtraFastBooster( trees_per_layer=trees_per_layer, max_depth=max_depth, nfea

booster.fit(Xte, Yt, Xve, Yv,  F,  FST, callbacks=[logger])

booster.save('saved_booster.npz')


pv = booster.predict(Xve, Yv.shape[0])


np.corrcoef(pv, Yv)
```

```
/usr/local/lib/python3.10/dist-packages/numba/cuda/dispatcher.py:536: NumbaPerformanceWarning: Grid size 127 will likely re
  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

```
@jit(nopython=True)
def pack_cuts_32(X, n):

  Xo = np.zeros( ( n*X.shape[0],  ( X.shape[1]+ 31 )>>5 )  , np.uint32)

  for f in range( X.shape[0] ):
    for k in range( X.shape[1] ):

      x = X[f, k]

      for s in range(n):

        Xo[n*f + s, k>>5] |=  ( 1 << ( k&31 ) ) * ( x > s )


  return Xo




@numba.jit(nopython=True)
def _advance_and_predict_offline(X, P, I, V,  L_old, L_new, tree_set, max_depth):

  for depth in range(max_depth):
    for tree_fold in range(I.shape[1]):
      for k in range(L_new.shape[-1]):

        leaf = ( L_old[tree_fold, depth-1, k] if depth > 0 else 0 )

        lo = leaf + (1<<depth) - 1

        li = I[tree_set, tree_fold, lo]

        x = ( X[li, k>>5] >> ( k&31  ) ) & 1

        leaf =  2*leaf + x

        if depth < L_new.shape[1]:
          L_new[tree_fold, depth, k] = leaf


        P[k] += V[tree_set, tree_fold, 2*lo + 1 + x ]

  return


def predict_offline(X, I, V, max_depth, n):
  P = np.zeros((n,), np.int32)
  L_old = np.zeros(( I.shape[1], max_depth-1, n), np.uint16)
  L_new = np.zeros(( I.shape[1], max_depth-1, n), np.uint16)


  for k in range(I.shape[0]):

    _advance_and_predict_offline(X, P, I, V, L_old, L_new, k, min(k+1, max_depth))

```

```
    L_new, L_old = L_old, L_new

  return P


class OfflineExtraFastBooster(object):


  def __init__(self, path):
    data = np.load(path)

    self.V = data['V']
    self.I = data['I']
    self.max_depth = int.bit_count(self.I.shape[-1])


  def predict(self, X, n=None):

    n = 32*X.shape[1] if n is None else n

    return predict_offline(X, self.I, self.V, self.max_depth, n)
```

Offline Prediction

Verifying offlne preds for one era:

```
off = OfflineExtraFastBooster('saved_booster.npz')

b = Ev==np.min(Ev)

Xve = Xv[b].T

Xve = pack_cuts_32(Xve, 4)

pvo = off.predict(Xve, b.sum() )

np.corrcoef(pvo, pv[b])
```

```
array([[1., 1.],
       [1., 1.]])
```

Upload Diagnostics

```
pv = pv-np.min(pv)

pv = pv/np.max(pv)

pv = pv*.98+.01


df_sub['prediction'] = pv[:df_sub.shape[0]]


df sub.to csv('predictions.csv')
```

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