kddCup2012

May 16, 2018

1 KDD Cup 2012, Track 2

1.1 Predict the click-through rate of ads given the query and user information.

1.2 https://www.kaggle.com/c/kddcup2012-track2#Description

2 Imports

In [ ]:
```
import sys
sys.path.append("/home/satluri/Downloads/fastai/courses/dl1/") # go to parent dir
```

In [ ]:
```
%matplotlib inline
%reload_ext autoreload
%autoreload 2
```

In [ ]:
```
from fastai.structured import *
from fastai.column_data import *
np.set_printoptions(threshold=50, edgeitems=20)

PATH='data/'
```

In [ ]:
```
from IPython.display import HTML
```

3 Load 20million Rows of Training Dataset

In [5]:
```
train = pd.read_csv(f'{PATH}track2/training.txt', low_memory=False, delimiter='\t',
                    nrows=20000000,
                    names=['Click', 'Impression', 'DisplayURL', 'AdID', 'AdvertiserID',
                           'Depth', 'Position', 'QueryID', 'KeywordID', 'TitleID', 'Descr
                           'UserID'])
```

In [6]:
```
train.head()
```

Out[6]:

	Click	Impression	DisplayURL	AdID	AdvertiserID	Depth	\
0	0	1	4298118681424644510	7686695	385	3	
1	0	1	4860571499428580850	21560664	37484	2	
2	0	1	9704320783495875564	21748480	36759	3	
3	0	1	13677630321509009335	3517124	23778	3	

```
4      0           1   3284760244799604489  20758093         34535      1

   Position  QueryID  KeywordID  TitleID  DescriptionID  UserID
0         3     1601       5521     7709            576  490234
1         2  2255103        317    48989          44771  490234
2         3  4532751      60721   685038          29681  490234
3         1     1601       2155     1207           1422  490234
4         1  4532751      77819   266618         222223  490234
```

In [7]: `display(DataFrameSummary(train).summary())`

```
                Click Impression   DisplayURL         AdID AdvertiserID  \
count             2e+07      2e+07        2e+07        2e+07        2e+07
mean          0.0514166    1.21818  9.74253e+18  1.60136e+07      22704.4
std            0.258976     1.3206  4.94553e+18  7.17242e+06      11616.2
min                   0          1  4.82437e+14  1.00003e+06           78
25%                   0          1  5.51113e+18  9.02744e+06        16005
50%                   0          1  1.06076e+19  2.02209e+07        23808
75%                   0          1  1.43404e+19  2.11639e+07        30736
max                 127       1019  1.84459e+19  2.22375e+07        39164
counts         20000000   20000000     20000000     20000000     20000000
uniques              60        230        23264       348442        14440
missing               0          0            0            0            0
missing_perc         0%         0%           0%           0%           0%
types           numeric    numeric      numeric      numeric      numeric

                 Depth  Position      QueryID    KeywordID      TitleID  \
count             2e+07     2e+07        2e+07        2e+07        2e+07
mean            1.90476   1.45584  2.72971e+06      27866.6       129602
std            0.695201  0.617746  5.62217e+06      86076.1       388084
min                   1         1            0            0            0
25%                   1         1          747          217          334
50%                   2         1        30876         2242         6048
75%                   2         2  1.82638e+06        15267        62770
max                   3         3  2.62436e+07  1.24978e+06  4.05142e+06
counts         20000000  20000000     20000000     20000000     20000000
uniques               3         3      3804596       502083      1321274
missing               0         0            0            0            0
missing_perc         0%        0%           0%           0%           0%
types           numeric   numeric      numeric      numeric      numeric

             DescriptionID       UserID
count                2e+07        2e+07
mean                 83634  4.74014e+06
std                 273534   5.7464e+06
min                      0            6
25%                    210       414926
50%                   2890  2.19189e+06
```

```
75%                  33296  7.15292e+06
max            3.17182e+06  2.39076e+07
counts            20000000     20000000
uniques            1051274      3941316
missing                  0            0
missing_perc            0%           0%
types               numeric      numeric
```

```
In [8]: # Convert 'Click' to Boolean
        mask = train.Click >= 1
        column_name = 'Click'
        train.loc[mask, column_name] = True
        mask = train.Click < 1
        train.loc[mask, column_name] = False
```

```
In [9]: display(DataFrameSummary(train).summary())
```

```
                Click Impression   DisplayURL         AdID AdvertiserID  \
count             NaN      2e+07        2e+07        2e+07        2e+07
mean              NaN    1.21818  9.74253e+18  1.60136e+07      22704.4
std               NaN     1.3206  4.94553e+18  7.17242e+06      11616.2
min               NaN          1  4.82437e+14  1.00003e+06           78
25%               NaN          1  5.51113e+18  9.02744e+06        16005
50%               NaN          1  1.06076e+19  2.02209e+07        23808
75%               NaN          1  1.43404e+19  2.11639e+07        30736
max               NaN       1019  1.84459e+19  2.22375e+07        39164
counts       20000000   20000000     20000000     20000000     20000000
uniques             2        230        23264       348442        14440
missing             0          0            0            0            0
missing_perc       0%         0%           0%           0%           0%
types            bool    numeric      numeric      numeric      numeric

                Depth  Position      QueryID    KeywordID      TitleID  \
count            2e+07     2e+07        2e+07        2e+07        2e+07
mean           1.90476   1.45584  2.72971e+06      27866.6       129602
std           0.695201  0.617746  5.62217e+06      86076.1       388084
min                  1         1            0            0            0
25%                  1         1          747          217          334
50%                  2         1        30876         2242         6048
75%                  2         2  1.82638e+06        15267        62770
max                  3         3  2.62436e+07  1.24978e+06  4.05142e+06
counts        20000000  20000000     20000000     20000000     20000000
uniques              3         3      3804596       502083      1321274
missing              0         0            0            0            0
missing_perc        0%        0%           0%           0%           0%
types          numeric   numeric      numeric      numeric      numeric
```

```
             DescriptionID       UserID
count                2e+07        2e+07
mean                 83634  4.74014e+06
std                 273534   5.7464e+06
min                      0            6
25%                    210       414926
50%                   2890  2.19189e+06
75%                  33296  7.15292e+06
max            3.17182e+06  2.39076e+07
counts            20000000     20000000
uniques            1051274      3941316
missing                  0            0
missing_perc            0%           0%
types              numeric      numeric
```

```
In [10]: cat_vars = ['AdID', 'AdvertiserID', 'QueryID', 'Depth', 'Position',
                     'KeywordID', 'TitleID', 'DescriptionID', 'UserID']
         #cont_vars = []

In [11]: #Now that we've engineered all our features, we need to convert to input compatible wit
         #This includes converting categorical variables into contiguous integers or one-hot enc
         #normalizing continuous features to standard normal, etc.

         for v in cat_vars: train[v] = train[v].astype('category').cat.as_ordered()
         #for v in cont_vars: train[v] = train[v].astype('float32')

In [12]: # Inspect cardinality of each categorical variable
         cat_sz = [(c, len(train[c].cat.categories)+1) for c in cat_vars]
         cat_sz

Out[12]: [('AdID', 348443),
          ('AdvertiserID', 14441),
          ('QueryID', 3804597),
          ('Depth', 4),
          ('Position', 4),
          ('KeywordID', 502084),
          ('TitleID', 1321275),
          ('DescriptionID', 1051275),
          ('UserID', 3941317)]

In [13]: # Create embeddings
         emb_szs = [(c, min(50, (c+1)//2)) for _,c in cat_sz]
         emb_szs

Out[13]: [(348443, 50),
          (14441, 50),
          (3804597, 50),
          (4, 2),
```

```
          (4, 2),
          (502084, 50),
          (1321275, 50),
          (1051275, 50),
          (3941317, 50)]
```

```
In [14]: train.reset_index(inplace=True)
```

```
In [15]: train, y, nas, mapper = proc_df(train, 'Click', do_scale=True, skip_flds=['Impression',
```

```
In [16]: train.head()
```

Out[16]:

	index	AdID	AdvertiserID	Depth	Position	QueryID	KeywordID	\
0	-1.732051	101756	117	3	3	1588	5521	
1	-1.732051	303367	13287	2	2	1025069	318	
2	-1.732050	309701	12693	3	3	1445183	60612	
3	-1.732050	33557	7272	3	1	1588	2155	
4	-1.732050	214400	11136	1	1	1445183	77618	

	TitleID	DescriptionID	UserID
0	7631	573	86666
1	48340	44268	86666
2	611350	29356	86666
3	1195	1405	86666
4	259865	217301	86666

```
In [17]: display(DataFrameSummary(train).summary())
```

	index	AdID	AdvertiserID	Depth	Position	\
count	2e+07	2e+07	2e+07	2e+07	2e+07	
mean	-7.30019e-17	179676	7216.65	1.90476	1.45584	
std	1	89741.3	3900.66	0.695201	0.617746	
min	-1.73205	1	1	1	1	
25%	-0.866025	116769	4801	1	1	
50%	0	172191	7302	2	1	
75%	0.866025	251032	9705	2	2	
max	1.73205	348442	14440	3	3	
counts	20000000	20000000	20000000	20000000	20000000	
uniques	20000000	348442	14440	3	3	
missing	0	0	0	0	0	
missing_perc	0%	0%	0%	0%	0%	
types	numeric	numeric	numeric	numeric	numeric	

	QueryID	KeywordID	TitleID	DescriptionID	UserID
count	2e+07	2e+07	2e+07	2e+07	2e+07
mean	600613	24675.8	97212.3	64112.6	799075
std	963068	62984.6	223124	161840	960937
min	1	1	1	1	1
25%	743	218	333	210	73539

50%	30697	2242	5981	2859	377732
75%	918997	15263	61908	32931	1.21622e+06
max	3.8046e+06	502083	1.32127e+06	1.05127e+06	3.94132e+06
counts	20000000	20000000	20000000	20000000	20000000
uniques	3804596	502083	1321274	1051274	3941316
missing	0	0	0	0	0
missing_perc	0%	0%	0%	0%	0%
types	numeric	numeric	numeric	numeric	numeric

4 DL

```
In [18]: def accuracy(y_pred, targ):
             correct = 0
             for i in range(len(y_pred)):
                 clicked = 1 if y_pred[i][0] >= 0.5 else 0
                 correct += 1 if clicked == targ[i][0] else 0
             return correct/len(y_pred)

In [19]: def inv_y(a): return np.exp(a)

         def exp_rmspe(y_pred, targ):
             targ = inv_y(targ)
             pct_var = (targ - inv_y(y_pred))/targ
             return math.sqrt((pct_var**2).mean())

In [20]: def intersection(lst1, lst2):

             # Use of hybrid method
             temp = set(lst2)
             lst3 = [value for value in lst1 if value in temp]
             return lst3
```

4.1 Train on Rows [0:10million]

```
In [21]: n = len(train)-1
         val_idxs = get_cv_idxs(n, val_pct=.1)
         val_idxs[0:5]

Out[21]: array([ 6323247, 18866225,   831796, 18686831,  3938314])

In [22]: batch_val_idxs = intersection(val_idxs[val_idxs>=0], val_idxs[val_idxs<1000000])

In [23]: batch_val_idxs.sort()

In [24]: (batch_val_idxs[0], batch_val_idxs[len(batch_val_idxs)-1])

Out[24]: (1, 999998)
```

```
In [25]: model = ColumnarModelData.from_data_frame(PATH, batch_val_idxs,
                                                   train[0:1000000], y[0:1000000].astype(np.floa
                                                   cat_vars, bs=4096)

In [26]: m = model.get_learner(emb_szs, len(train.columns)-len(cat_vars),
                               0.04, 1, [1000,500], [0.001,0.01], y_range=(0,1))

In [27]: m.crit = F.binary_cross_entropy
         m.crit

Out[27]: <function torch.nn.functional.binary_cross_entropy>

In [28]: m.lr_find()

HBox(children=(IntProgress(value=0, description='Epoch', max=1), HTML(value='')))


  0%|          | 1/220 [00:01<06:04,  1.66s/it, loss=0.708]



        ---------------------------------------------------------------------------

        RuntimeError                              Traceback (most recent call last)

        <ipython-input-28-0510517b8367> in <module>()
    ----> 1 m.lr_find()


        ~/Downloads/fastai/courses/dl1/fastai/learner.py in lr_find(self, start_lr, end_lr, wds,
        328         layer_opt = self.get_layer_opt(start_lr, wds)
        329         self.sched = LR_Finder(layer_opt, len(self.data.trn_dl), end_lr, linear=line
    --> 330         self.fit_gen(self.model, self.data, layer_opt, 1, **kwargs)
        331         self.load('tmp')
        332


        ~/Downloads/fastai/courses/dl1/fastai/learner.py in fit_gen(self, model, data, layer_opt
        232             metrics=metrics, callbacks=callbacks, reg_fn=self.reg_fn, clip=self.clip
        233             swa_model=self.swa_model if use_swa else None, swa_start=swa_start,
    --> 234             swa_eval_freq=swa_eval_freq, **kwargs)
        235
        236     def get_layer_groups(self): return self.models.get_layer_groups()


        ~/Downloads/fastai/courses/dl1/fastai/model.py in fit(model, data, n_epochs, opt, crit,
        127             batch_num += 1
        128             for cb in callbacks: cb.on_batch_begin()
    --> 129             loss = model_stepper.step(V(x),V(y), epoch)
```

```
    130             avg_loss = avg_loss * avg_mom + loss * (1-avg_mom)
    131             debias_loss = avg_loss / (1 - avg_mom**batch_num)


    ~/Downloads/fastai/courses/dl1/fastai/model.py in step(self, xs, y, epoch)
     53         if self.loss_scale != 1: assert(self.fp16); loss = loss*self.loss_scale
     54         if self.reg_fn: loss = self.reg_fn(output, xtra, raw_loss)
---> 55         loss.backward()
     56         if self.fp16: update_fp32_grads(self.fp32_params, self.m)
     57         if self.loss_scale != 1:


    ~/anaconda3/lib/python3.6/site-packages/torch/autograd/variable.py in backward(self, gra
    165                 Variable.
    166         """
--> 167         torch.autograd.backward(self, gradient, retain_graph, create_graph, retain_v
    168
    169     def register_hook(self, hook):


    ~/anaconda3/lib/python3.6/site-packages/torch/autograd/__init__.py in backward(variables
     97
     98     Variable._execution_engine.run_backward(
---> 99         variables, grad_variables, retain_graph)
    100
    101


    ~/anaconda3/lib/python3.6/site-packages/torch/autograd/function.py in apply(self, *args)
     89
     90     def apply(self, *args):
---> 91         return self._forward_cls.backward(self, *args)
     92
     93


    ~/anaconda3/lib/python3.6/site-packages/torch/autograd/function.py in wrapper(ctx, *args
    203         tensor_args = [arg.data if isinstance(arg, Variable) else arg
    204                        for arg in args]
--> 205         outputs = fn(ctx, *tensor_args)
    206         # XXX: this is only an approximation of these flags - there's no way
    207         # to figure out if fn didn't use ctx.saved_variables and as a result


    ~/anaconda3/lib/python3.6/site-packages/torch/nn/_functions/thnn/sparse.py in backward(c
     84                     _sorted = _indices = None
     85
---> 86             grad_weight = grad_output.new(ctx._weight_size).zero_()
```

```
 87             # Doesn't support Variable grad_output
 88             ctx._backend.LookupTable_accGradParameters(


RuntimeError: cuda runtime error (2) : out of memory at /opt/conda/conda-bld/pytorch_151
```

In [ ]:
```
%matplotlib inline
m.sched.plot()
```

In [ ]:
```
lr=3e-5
```

In [ ]:
```
m.fit(lr, 1, metrics=[accuracy, exp_rmspe])
```

In [ ]:
```
m.save('kddCup2012')
```

In [ ]:
```
m.load('kddCup2012')
```

In [ ]:
```
m.fit(lr, 1, metrics=[accuracy, exp_rmspe])
```

In [ ]:
```
list(m.data.trn_ds)
```

In [ ]:
```
preds = m.predict()#is_test=True)
```

In [ ]:
```
#preds
```

In [ ]:
```
# Number of predicted Clicks in the validation set
len(preds[preds>=0.5])
```

In [ ]:
```
# Actual number of Clicks in the validation set
actualClicks = 0
for click in y[val_idxs]:
    actualClicks += 1 if click == 1 else 0
actualClicks
```