Recommender System

This program uses colaborative fitering to recommend movies

Thr program uses fastai libraries

In [1]:

```
# Import relevant Libraries

%reload_ext autoreload
%autoreload 2
%matplotlib inline

from fastai.learner import *
from fastai.column_data import *
```

```
home/paperspace/anaconda3/envs/fastai/lib/python3.6/site-packages/sklearn/e
semble/weight_boosting.py:29: DeprecationWarning: numpy.core.umath_tests is
n internal NumPy module and should not be imported. It will be removed in a
uture NumPy release.
  from numpy.core.umath_tests import inner1d
```

In [2]:

```
import os
#Change working Directory
os.chdir('/home/paperspace/fastai/courses/SelfCodes/Colaborative_Fiter_IMDB/data')
%pwd

path='/home/paperspace/fastai/courses/SelfCodes/Colaborative_Fiter_IMDB/data/ml-latest-smal
```

In [3]:

```
# Get Data for model development

# ! wget http://files.grouplens.org/datasets/movielens/ml-latest-small.zip
```

In [4]:

```
# UNZIP ml-latest-small.zip
#import zipfile
#with zipfile.ZipFile("/home/paperspace/fastai/courses/SelfCodes/Colaborative_Fiter_IMDB/da
# zip_ref.extractall("/home/paperspace/fastai/courses/SelfCodes/Colaborative_Fiter_IMDB/dat
```

We're working with the movielens data, which contains one rating per row:

In [5]:

```
ratings = pd.read_csv(path+'ratings.csv')
ratings.head()
```

Out[5]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Just for display purposes, let's read in the movie names too.

In [6]:

```
movies = pd.read_csv(path+'movies.csv')
movies.head()
```

Out[6]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Collaborative filtering

In [7]:

```
# create a validation set by picking random set of ID's.
# wd is a weight decay for L2 regularization,
# and n_factors is how big an embedding matrix we want.

val_idxs = get_cv_idxs(len(ratings))
wd=2e-4
n_factors = 50
```

Create a model data object from CSV file

In [8]:

```
cf = CollabFilterDataset.from_csv(path, 'ratings.csv', 'userId', 'movieId', 'rating')
```

Get a learner that is suitable for the model data, and fit the model:

In [9]:

```
learn = cf.get_learner(n_factors, val_idxs, 64, opt_fn=optim.Adam)
learn.fit(1e-2, 2, wds=wd, cycle_len=1, cycle_mult=2)
```

Epoch 100% 3/3 [01:37<00:00, 31.02s/it]

```
epoch      trn_loss   val_loss
    0      0.746956   0.772499
    1      0.711768   0.750826
    2      0.590427   0.735018
```

Out[9]:

```
[array([0.73502])]
```

Since the output is Mean Squared Error, you can take RMSE by:

In [10]:

```
math.sqrt(0.765)
```

Out[10]:

```
0.8746427842267951
```

Looking good - we've found a solution better than any of those benchmarks! Let's take a look at how the predictions compare to actuals for this model

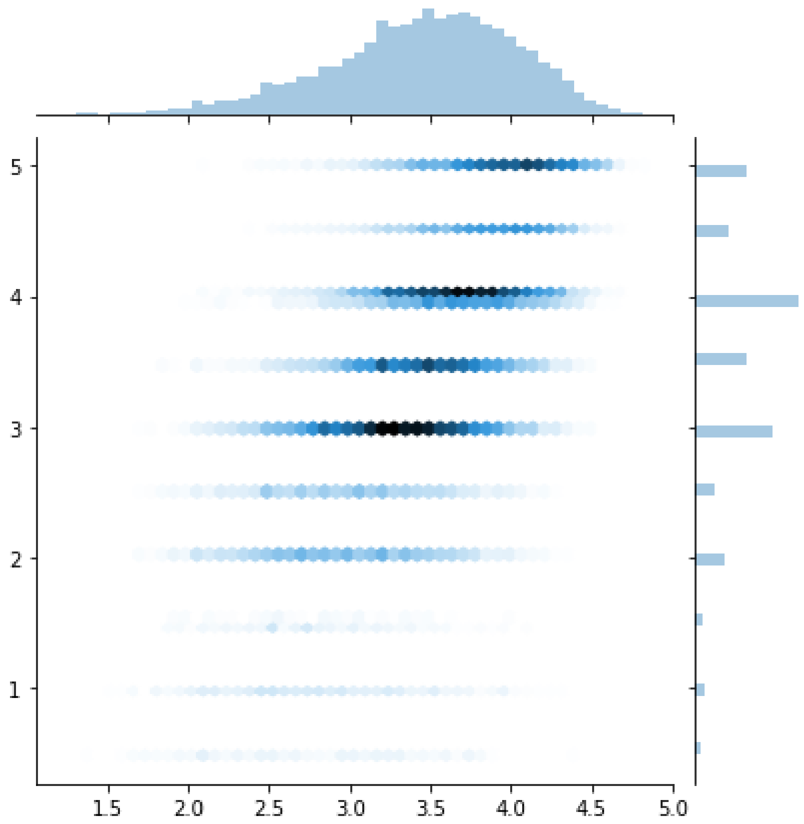
In [11]:

```
preds = learn.predict()
```

you can also plot using seaborn sns (built on top of matplotlib):

In [12]:

```
y=learn.data.val_y
sns.jointplot(preds, y, kind='hex', stat_func=None);
```



Collab filtering from scratch

Dot product example

In [13]:

```
a = T([[1.,2],[3,4]])
b = T([[2.,2],[10,10]])
a,b
```

Out[13]:

```
(
  1  2
  3  4
 [torch.FloatTensor of size 2x2],
   2   2
  10  10
 [torch.FloatTensor of size 2x2])
```

In [14]:

```
# element-wise assuming that they both have the same dimensionality.
a*b
```

Out[14]:

```
  2   4
 30  40
[torch.FloatTensor of size 2x2]
```

In [15]:

```
# The below is how you would calculate the dot product of two vectors (e.g. (1, 2)·(2, 2) =
(a*b).sum(1)
```

Out[15]:

```
  6
 70
[torch.FloatTensor of size 2]
```

Building our first custom layer (i.e. PyTorch module)

In [16]:

```
class DotProduct (nn.Module):
   def forward(self, u, m): return (u*m).sum(1)
```

Now we can call it and get the expected result (notice that we do not need to say model.forward(a, b) to call the forward function)

In [17]:

```
model = DotProduct()
model(a,b)
```

Out[17]:

```
  6
 70
[torch.FloatTensor of size 2]
```

Building more complex module

This implementation has two additions to the DotProduct class:

Two nn.Embedding matrices Look up our users and movies in above embedding matrices

It is quite possible that user ID's are not contiguous which makes it hard to use as an index of embedding matrix.

So we will start by creating indexes that starts from zero and contiguous and replace ratings.userId column with the index by using Panda's apply function with an anonymous function lambda and do the same for ratings.movieId .

In [18]:

```
u_uniq = ratings.userId.unique()
user2idx = {o:i for i,o in enumerate(u_uniq)}
ratings.userId = ratings.userId.apply(lambda x: user2idx[x])

m_uniq = ratings.movieId.unique()
movie2idx = {o:i for i,o in enumerate(m_uniq)}
ratings.movieId = ratings.movieId.apply(lambda x: movie2idx[x])

n_users=int(ratings.userId.nunique())
n_movies=int(ratings.movieId.nunique())

print(n_users)
print(n_movies)
```

```
610
9724
```

In [19]:

```
# __init__ is a constructor which is now needed because our class needs to keep track of “s
# (how many movies, mow many users, how many factors, etc).

#  We initialized the weights to random numbers between 0 and 0.05

class EmbeddingDot(nn.Module):
    def __init__(self, n_users, n_movies):
        super().__init__()
        self.u = nn.Embedding(n_users, n_factors)
        self.m = nn.Embedding(n_movies, n_factors)
        self.u.weight.data.uniform_(0,0.05)
        self.m.weight.data.uniform_(0,0.05)


    def forward(self, cats, conts):
        users,movies = cats[:,0],cats[:,1]
        u,m = self.u(users),self.m(movies)
        return (u*m).sum(1)
```

Embedding is not a tensor but a variable. A variable does the exact same operations as a tensor but it also does automatic differentiation. To pull a tensor out of a variable, call data attribute. All the tensor functions have a variation with trailing underscore (e.g. uniform_) will do things in-place.

In [20]:

```
# define input and output from ratings table

x = ratings.drop(['rating', 'timestamp'],axis=1)
y = ratings['rating'].astype(np.float32)
```

We are reusing ColumnarModelData (from fast.ai library) , and that is the reason behind why there are both categorical and continuous variables in def forward(self, cats, conts) function in EmbeddingDot class . Since we do not have continuous variable in this case, we will ignore conts and use the first and second columns of cats as users and movies

In [21]:

```
data = ColumnarModelData.from_data_frame(path, val_idxs, x, y, ['userId', 'movieId'], 64)
```

In [26]:

```
! sudo apt update && sudo apt upgrade
```

```
it:1 http://archive.ubuntu.com/ubuntu (http://archive.ubuntu.com/ubuntu) xe
ial InRelease
it:2 http://archive.ubuntu.com/ubuntu (http://archive.ubuntu.com/ubuntu) xe
ial-updates InRelease
it:3 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu (http://ppa.launc
pad.net/graphics-drivers/ppa/ubuntu) xenial InRelease
it:4 http://archive.ubuntu.com/ubuntu (http://archive.ubuntu.com/ubuntu) xe
ial-backports InRelease
it:5 http://security.ubuntu.com/ubuntu (http://security.ubuntu.com/ubuntu)
xenial-security InRelease
gn:6 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86
_64 (http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_
4)  InRelease
it:7 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86
_64 (http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_
4)  Release
it:8 http://repo.saltstack.com/apt/ubuntu/16.04/amd64/latest (http://repo.s
ltstack.com/apt/ubuntu/16.04/amd64/latest) xenial InRelease
eading package lists... Done
uilding dependency tree
eading state information... Done
 packages can be upgraded. Run 'apt list --upgradable' to see them.
eading package lists... Done
uilding dependency tree
eading state information... Done
alculating upgrade... Done
he following packages were automatically installed and are no longer requir
d:
 libllvm5.0 libqmi-glib1
se 'sudo apt autoremove' to remove them.
he following packages have been kept back:
 cuda cuda-drivers
 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
```

In [27]:

```
# NVidia GPU with programming framework CUDA is critical & following command must return tr
torch.cuda.is_available()
```

Out[27]:

```
False
```

In [24]:

```
# Make sure deep learning package from CUDA CuDNN is enabled for improving training perform
torch.backends.cudnn.enabled
```

Out[24]:

```
True
```

In [25]:

```
wd=1e-5
model = EmbeddingDot(n_users, n_movies).cuda()
opt = optim.SGD(model.parameters(), 1e-1, weight_decay=wd, momentum=0.9)
```

```
---------------------------------------------------------------------------
 untimeError                               Traceback (most recent call last)
 ipython-input-25-b665a9595d7f> in <module>
      1 wd=1e-5
----> 2 model = EmbeddingDot(n_users, n_movies).cuda()
      3 opt = optim.SGD(model.parameters(), 1e-1, weight_decay=wd, momentum=
 .9)

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/nn/modules/module.
 y in cuda(self, device)
    214             Module: self
    215         """
--> 216         return self._apply(lambda t: t.cuda(device))
    217
    218     def cpu(self):

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/nn/modules/module.
 y in _apply(self, fn)
    144     def _apply(self, fn):
    145         for module in self.children():
--> 146             module._apply(fn)
    147
    148         for param in self._parameters.values():

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/nn/modules/module.
 y in _apply(self, fn)
    150                 # Variables stored in modules are graph leaves, and
 we don't
    151                 # want to create copy nodes, so we have to unpack th
  data.
--> 152                 param.data = fn(param.data)
    153                 if param._grad is not None:
    154                     param._grad.data = fn(param._grad.data)

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/nn/modules/module.
 y in <lambda>(t)
    214             Module: self
    215         """
--> 216         return self._apply(lambda t: t.cuda(device))
    217
    218     def cpu(self):

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/_utils.py in _cuda
 self, device, async)
     67         else:
     68             new_type = getattr(torch.cuda, self.__class__.__name__)
---> 69             return new_type(self.size()).copy_(self, async)
     70
     71

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/cuda/__init__.py i
  _lazy_new(cls, *args, **kwargs)
    382 @staticmethod
    383 def _lazy_new(cls, *args, **kwargs):
```

```
--> 384     _lazy_init()
    385     # We need this method only for lazy init, so we can remove it
    386     del _CudaBase.__new__

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/cuda/__init__.py i
  _lazy_init()
    140             "Cannot re-initialize CUDA in forked subprocess. " + ms
 )
    141     _check_driver()
--> 142     torch._C._cuda_init()
    143     torch._C._cuda_sparse_init()
    144     _cudart = _load_cudart()

 untimeError: cuda runtime error (38) : no CUDA-capable device is detected a
  /opt/conda/conda-bld/pytorch_1518244421288/work/torch/lib/THC/THCGeneral.
 :70
```

Optim is what gives us the optimizers in PyTorch. model.parameters() is one of the function inherited from nn.Modules that gives us all the weight to be updated/learned.

In [ ]:

```
fit(model, data, 3, opt, F.mse_loss)
```

In [ ]:

```
set_lrs(opt, 0.01)
```

In [ ]:

```
fit(model, data, 3, opt, F.mse_loss)
```

Improve our model

Bias—to adjust to generally popular movies or generally enthusiastic users.

In [ ]:

```
min_rating,max_rating = ratings.rating.min(),ratings.rating.max()
min_rating,max_rating
```

In [ ]:

```
def get_emb(ni,nf):
    e = nn.Embedding(ni, nf)
    e.weight.data.uniform_(-0.01,0.01)
    return e

class EmbeddingDotBias(nn.Module):
    def __init__(self, n_users, n_movies):
        super().__init__()
        (self.u, self.m, self.ub, self.mb) = [get_emb(*o) for o in [
            (n_users, n_factors), (n_movies, n_factors), (n_users,1), (n_movies,1)
        ]]

    def forward(self, cats, conts):
        users,movies = cats[:,0],cats[:,1]
        um = (self.u(users)* self.m(movies)).sum(1)
        res = um + self.ub(users).squeeze() + self.mb(movies).squeeze()
        res = F.sigmoid(res) * (max_rating-min_rating) + min_rating
        return res.view(-1, 1)
```

In [ ]:

```
wd=2e-4
model = EmbeddingDotBias(cf.n_users, cf.n_items).cuda()
opt = optim.SGD(model.parameters(), 1e-1, weight_decay=wd, momentum=0.9)
```

Can we squish the ratings so that it is between 1 and 5? Yes! By putting the prediction through sigmoid function will result in number between 1 and 0. So in our case, we can multiply that by 4 and add 1—which will result in number between 1 and 5.

In [ ]:

```
fit(model, data, 3, opt, F.mse_loss)
```

In [ ]:

```
set_lrs(opt, 1e-2)
```

In [ ]:

```
fit(model, data, 3, opt, F.mse_loss)
```

Mini Net

Rather than calculating the dot product of user embedding vector and movie embedding vector to get a prediction, we will concatenate the two and feed it through neural net.

In [ ]:

```
class EmbeddingNet(nn.Module):
    def __init__(self, n_users, n_movies, nh=10, p1=0.05, p2=0.5):
        super().__init__()
        (self.u, self.m) = [get_emb(*o) for o in [
            (n_users, n_factors), (n_movies, n_factors)]]
        self.lin1 = nn.Linear(n_factors*2, nh)
        self.lin2 = nn.Linear(nh, 1)
        self.drop1 = nn.Dropout(p1)
        self.drop2 = nn.Dropout(p2)

    def forward(self, cats, conts):
        users,movies = cats[:,0],cats[:,1]
        x = self.drop1(torch.cat([self.u(users),self.m(movies)], dim=1))
        x = self.drop2(F.relu(self.lin1(x)))
        return F.sigmoid(self.lin2(x)) * (max_rating-min_rating+1) + min_rating-0.5
```

Notice that we no longer has bias terms since Linear layer in PyTorch already has a build in bias. nh is a number of activations a linear layer creates

In [ ]:

```
wd=1e-5
model = EmbeddingNet(n_users, n_movies).cuda()
opt = optim.Adam(model.parameters(), 1e-3, weight_decay=wd)
```

It only has one hidden layer, so maybe not “deep”, but this is definitely a neural network.

Now that we have neural net, there are many things we can try:

Add dropouts

Use different embedding sizes for user embedding and movie embedding

Not only user and movie embeddings, but append movie genre embedding and/or timestamp from the original data.

Increase/decrease number of hidden layers and activations

Increase/decrease regularization

In [ ]:

```
fit(model, data, 3, opt, F.mse_loss)
```

In [ ]:

```
set_lrs(opt, 1e-3)
```