DATA MINING

DIGITAL ASSIGNMENT 6

ABHIRUPA MITRA - 17BCE0437

Recommendar Systems

Implement a recommender system using TensorFlow and Cloud Machine Learning Engine in Google Cloud Platform.

Objectives:

- Prepare Google Analytics data from BigQuery for training a recommendation model.
- Train the recommendation model.
- Tune model hyperparameters for Google Analytics data.
- Run the TensorFlow model code on Google Analytics data to generate recommendations.

Submit the step-by-step docs for implementing a recommendation system on GCP.

STEP 1: Create a browser based terminal and securely connect to the VM Instance:

```
sudo apt-get update

sudo apt-get install -y git bzip2

git clone https://github.com/GoogleCloudPlatform/tensorflow-recommendation-wals

wget https://repo.continuum.io/miniconda/Miniconda2-latest-Linux-x86_64.sh
bash Miniconda2-latest-Linux-x86_64.sh

cd tensorflow-recommendation-wals
conda create -n tfrec
conda install -n tfrec --file conda.txt
source activate tfrec
pip install -r requirements.txt
pip install tensorflow

curl -O 'http://files.grouplens.org/datasets/movielens/ml-100k.zip'
unzip ml-100k.zip
mkdir -p data
cp ml-100k/u.data data/

curl -O 'http://files.grouplens.org/datasets/movielens/ml-1m.zip'
unzip ml-1m.zip
mkdir -p data
cp ml-1m/ratings.dat data/

curl -O 'http://files.grouplens.org/datasets/movielens/ml-20m.zip'
unzip ml-20m.zip
mkdir -p data
cp ml-20m/ratings.csv data/
```

STEP 2: Model Code:

In [2]:

```python
import pandas as pd
ratings_df = pd.read_csv(input_file,
                         sep=args['delimiter'],
                         names=headers,
                         header=header_row,
                         dtype={
                           'user_id': np.int32,
                           'item_id': np.int32,
                           'rating': np.float32,
                           'timestamp': np.int32,
                         })
ratings = ratings_df.as_matrix(['user_id', 'item_id', 'rating'])
# deal with 1-based user indices
ratings[:,0] -= 1
ratings[:,1] -= 1

np_items = ratings_df.item_id.as_matrix()
unique_items = np.unique(np_items)
n_items = unique_items.shape[0]
max_item = unique_items[-1]

# map unique items down to an array 0..n_items-1
z = np.zeros(max_item+1, dtype=int)
z[unique_items] = np.arange(n_items)
i_r = z[np_items]

test_set_size = len(ratings) / TEST_SET_RATIO
test_set_idx = np.random.choice(xrange(len(ratings)),
                                size=test_set_size, replace=False)
test_set_idx = sorted(test_set_idx)
```

```python
ts_ratings = ratings[test_set_idx]
tr_ratings = np.delete(ratings, test_set_idx, axis=0)

u_tr, i_tr, r_tr = zip(*tr_ratings)
tr_sparse = coo_matrix((r_tr, (u_tr, i_tr)), shape=(n_users, n_items))

# Implementing the WALS algorithm in TensorFlow
input_tensor = tf.SparseTensor(indices=zip(data.row, data.col),
                                values=(data.data).astype(np.float32),
                                dense_shape=data.shape)
model = factorization_ops.WALSModel(num_rows, num_cols, dim,
                                    unobserved_weight=unobs,
                                    regularization=reg,
                                    row_weights=row_wts,
                                    col_weights=col_wts)
# retrieve the row and column factors
row_factor = model.row_factors[0]
col_factor = model.col_factors[0]

row_update_op = model.update_row_factors(sp_input=input_tensor)[1]
col_update_op = model.update_col_factors(sp_input=input_tensor)[1]

sess.run(model.initialize_op)
sess.run(model.worker_init)
for _ in xrange(num_iterations):
    sess.run(model.row_update_prep_gramian_op)
    sess.run(model.initialize_row_update_op)
    sess.run(row_update_op)
    sess.run(model.col_update_prep_gramian_op)
    sess.run(model.initialize_col_update_op)
    sess.run(col_update_op)

#  evaluate output factor matrices
output_row = row_factor.eval(session=session)
output_col = col_factor.eval(session=session)
```

Step3: Train the model locally

Training the model locally is useful for development purposes. It allows you to rapidly test code changes and to include breakpoints for easy debugging. To run the model on Cloud Shell or from your local system, run the `mltrain.sh` script from the `wals_ml_engine` directory using the `local` option.

```
cd wals_ml_engine
./mltrain.sh local ../data u.data
./mltrain.sh local ../data ratings.csv --headers --delimiter ,
```

OUTPUT:

```
INFO:tensorflow:Train Start: <timestamp>
...
INFO:tensorflow:Train Finish: <timestamp>
INFO:tensorflow:train RMSE = 1.29
INFO:tensorflow:test RMSE = 1.34
```

```
BUCKET=gs://[YOUR_BUCKET_NAME]
gsutil cp -r data/u.data $BUCKET/data/u.data
gsutil cp -r data/ratings.dat $BUCKET/data/ratings.dat
gsutil cp -r data/ratings.csv $BUCKET/data/ratings.csv

cd wals_ml_engine
./mltrain.sh train ${BUCKET} data/u.data
./mltrain.sh train ${BUCKET} data/ratings.dat --delimiter ::
```

Step 4: Hyper parameters to be tuned:

Hyperparameter name and description	Default Value	Scale
`latent_factors` Number of latent factors K	5	UNIT_REVERSE_LOG_SCALE
`regularization` L2 Regularization constant	0.07	UNIT_REVERSE_LOG_SCALE
`unobs_weight` Weight on unobserved ratings matrix entries	0.01	UNIT_REVERSE_LOG_SCALE
`feature_wt_factor` Weight on observed entries	130	UNIT_LINEAR_SCALE
`feature_wt_exp` Feature weight exponent	1	UNIT_LOG_SCALE
`num_iters` Number of alternating least squares iterations	20	UNIT_LINEAR_SCALE

Table 1. Hyperparameter names and default values used in the model

The standard_gpu machine type is specified in the scaleTier parameter, so tuning takes place on a GPU-provisioned machine. The configuration file looks like this:

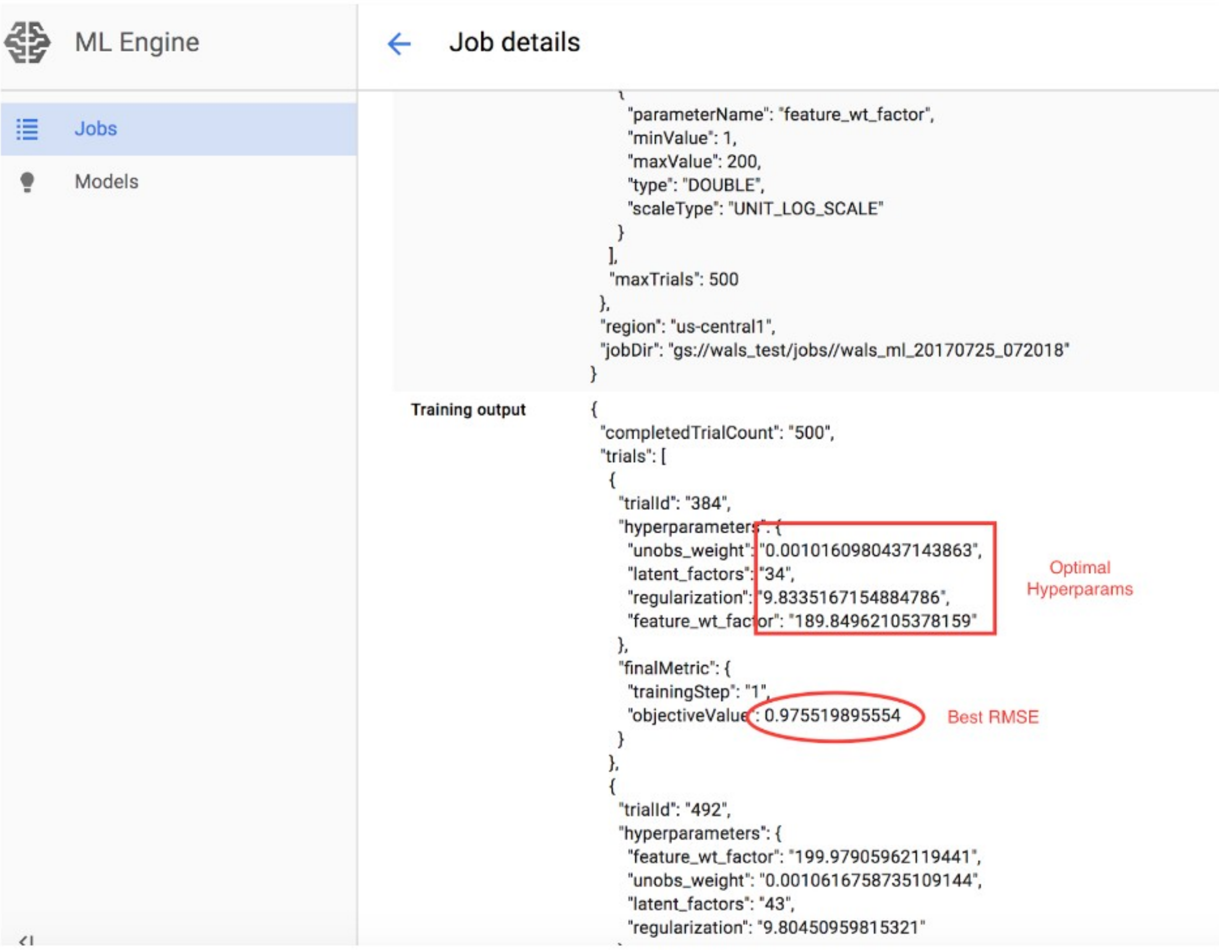
```
{
    "trainingInput": {
    "scaleTier": "CUSTOM",
    "masterType": "standard_gpu",
    "hyperparameters": {
        "goal": "MINIMIZE",
        "params": [
            {
            "parameterName": "regularization",
            "type": "DOUBLE",
```

```
          "minValue": "0.001",
          "maxValue": "10.0",
          "scaleType": "UNIT_REVERSE_LOG_SCALE"
          },
          {
          "parameterName": "latent_factors",
          "type": "INTEGER",
          "minValue": "5",
          "maxValue": "50",
          "scaleType": "UNIT_REVERSE_LOG_SCALE"
          },
          {
          "parameterName": "unobs_weight",
          "type": "DOUBLE",
          "minValue": "0.001",
          "maxValue": "5.0",
          "scaleType": "UNIT_REVERSE_LOG_SCALE"
          },
          {
          "parameterName": "feature_wt_factor",
          "type": "DOUBLE",
          "minValue": "1",
          "maxValue": "200",
          "scaleType": "UNIT_LOG_SCALE"
          }
     ],
     "maxTrials": 500
}
}
}
```

```
summary = Summary(value=[Summary.Value(tag='training/hptuning/metric',
                                       simple_value=metric)])

eval_path = os.path.join(args['output_dir'], 'eval')
summary_writer = tf.summary.FileWriter(eval_path)

# Note: adding the summary to the writer is enough for hyperparam tuning.
# The ml engine system is looking for any summary added with the
# hyperparam metric tag.
summary_writer.add_summary(summary)

if args.hypertune:
  # if tuning, join the trial number to the output path
  trial = json.loads(os.environ.get('TF_CONFIG', '{}')).get('task',
{}).get('trial', '')
  output_dir = os.path.join(job_dir, trial)
else:
  output_dir = os.path.join(job_dir, args.job_name)
```

Step 5 :Running the hyperparameter tuning job:

```
./mltrain.sh tune $BUCKET data/u.data
```



Hyperparameter Name	Description	Value From Tuning
`latent_factors`	Latent factors K	34
`regularization`	L2 Regularization constant	9.83
`unobs_weight`	Unobserved weight	0.001
`feature_wt_factor`	Observed weight	189.8
`feature_wt_exp`	Feature weight exponent	N/A
`num_iters`	Number of iterations	N/A

Table 2. Values discovered by Cloud ML Engine hyperparameter tuning

Dataset	RMSE with default hyperparameters	RMSE after hyperparameter tuning
100k	1.06	0.98
1m	1.11	0.90
20m	1.30	0.88

Table 3. Summary of RMSE values on the test set for the different `MovieLens` datasets, before and after hyperparameter tuning

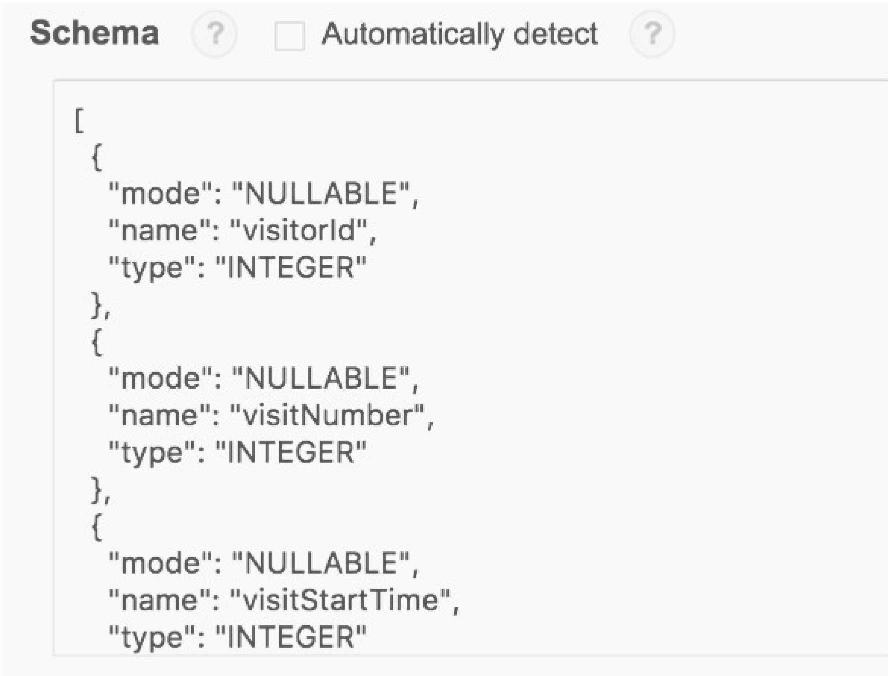
Step 6: Preparing the data from BigQuery for training:

Shell Commands:

```
BUCKET=gs://[bucket1]
gsutil cp
gs://solutions-public-assets/recommendation-tensorflow/data/ga_ses
sions_sample.json.gz ${BUCKET}/data/ga_sessions_sample.json.gz
```

For Location, select Google Cloud Storage, and then enter the following path:

`[bucket1]/data/ga_sessions_sample.json.gz`



Step 7:Exporting the training Data:

```
#legacySql
SELECT
 fullVisitorId as clientId,
 ArticleID as contentId,
 (nextTime - hits.time) as timeOnPage,
FROM(
  SELECT
    fullVisitorId,
    hits.time,
```

```
    MAX(IF(hits.customDimensions.index=10,
           hits.customDimensions.value,NULL)) WITHIN hits AS ArticleID,
    LEAD(hits.time, 1) OVER (PARTITION BY fullVisitorId, visitNumber
  ORDER BY hits.time ASC) as nextTime
  FROM [GA360_sample.ga_sessions_sample]
  WHERE hits.type = "PAGE"
) HAVING timeOnPage is not null and contentId is not null;
```

Step 8: Training the recommendation model:

```
BUCKET="gs://[YOUR_BUCKET_NAME]"
gs://[YOUR_BUCKET_NAME]/ga_pageviews.csv
./mltrain.sh train $BUCKET ga_pageviews.csv --data-type
web_views
```

The path for the job directory is created using the BUCKET argument passed to the mltrain.sh script, then /jobs/, and then the identifier of the training job. The job identifier is set in the mltrain.sh script as well. By default, that identifier is wals_ml_train appended with the job start date and time. For example, if you specified a BUCKET of gs://my_bucket, the model files would be saved to paths like these:

```
gs://my_bucket/jobs/wals_ml_train_20171201_120001/model/row.npy
gs://my_bucket/jobs/wals_ml_train_20171201_120001/model/col.npy
gs://my_bucket/jobs/wals_ml_train_20171201_120001/model/user.npy
gs://my_bucket/jobs/wals_ml_train_20171201_120001/model/item.npy
```

Step 8: Tuning model hyperparameters for Google Analytics data:

```
./mltrain.sh tune gs://your_bucket data/ga_pageviews.csv --data-
type web_views
```

Hyperparameter Name	Description	Value From Tuning
latent_factors	Latent factors K	30
regularization	L2 Regularization constant	5.05
unobs_weight	Unobserved weight	0.01
feature_wt_factor	Observed weight (linear)	N/A
feature_wt_exp	Feature weight exponent	5.05

Table 1 Values discovered by Cloud ML Engine hyperparameter tuning for the sample Google Analytics data

Step 10: Running Model Code To Generate Recommendations:

```
import numpy as np
from model import generate_recommendations

client_id = 1000163602560555666
already_rated = [295436355, 295044773, 295195092]
k = 5
user_map = np.load("/tmp/model/user.npy")
item_map = np.load("/tmp/model/item.npy")
row_factor = np.load("/tmp/model/row.npy")
col_factor = np.load("/tmp/model/col.npy")
user_idx = np.searchsorted(user_map, client_id)
user_rated = [np.searchsorted(item_map, i) for i in
already_rated]

recommendations = generate_recommendations(user_idx,
user_rated, row_factor, col_factor, k)

article_recommendations = [item_map[i] for i in
recommendations]
```

Generate recommendations in production:

The remaining components needed for a production system to serve recommendations on Google Analytics data include:

- Training a recommendation model on a regular schedule—for example, nightly.
- Serving the recommendations using an API.