VWFS Machine Learning Workshop

Recommender Systems

Ahmed Rashed - UHi

University of Hildesheim (UHi), Germany

March 20, 2019

Outline

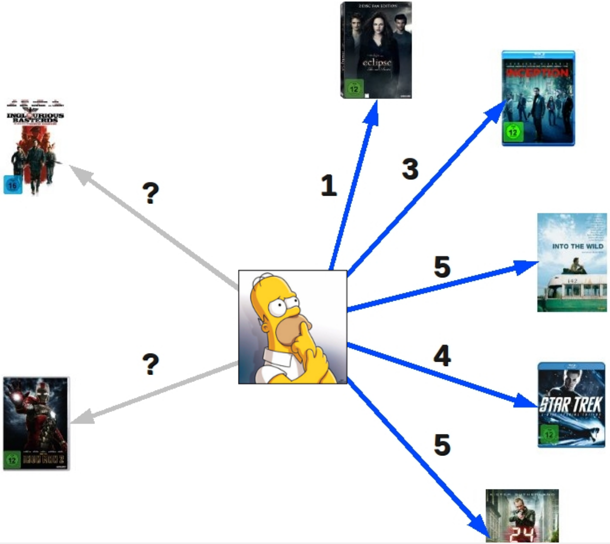
Outline

Why Recommender Systems?

- Powerful method for enabling users to filter large amounts of information
- Personalized recommendations can boost the revenue of an e-commerce system:
  - Amazon recommender systems
  - Netflix challgenge: 1 million dollars for improving their system on 10%
- Different applications:
  - E-commerce
  - Education
  - ...

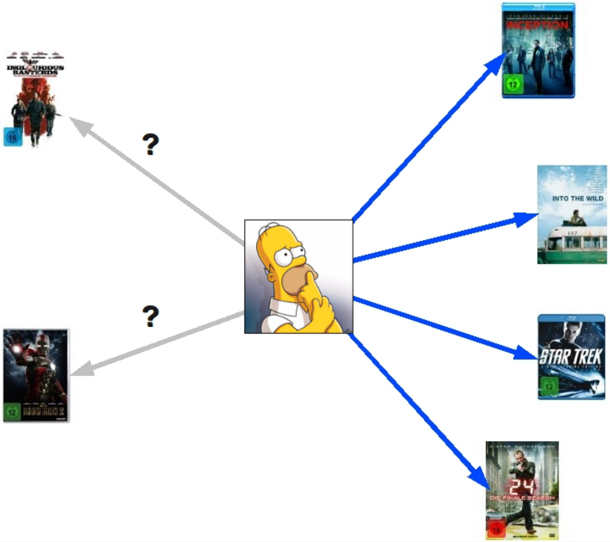
Prediction Version - Rating Prediction

Given the previously rated items, how the user will evaluate other items?



Ranking Version - Item Prediction

Which will be the next items to be consumed by a user?

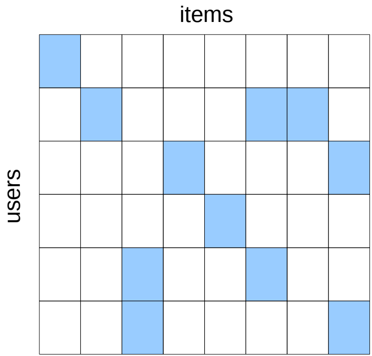


Outline

Formalization

- U - Set of Users
- I - Set of Items
- Ratings data $D \subseteq U \times I \times \mathbb{R}$

Rating data D are typically represented as a sparse matrix $\mathbf{R} \in \mathbb{R}^{|U| \times |I|}$



Recommender Systems - Task

Given a set of users U, items I and training data $D^{train} \subseteq U \times I \times \mathbb{R}$, find a function

$$\hat{r} : U \times I \rightarrow \mathbb{R}$$

That minimize the loss

$$error(\hat{r}, D^{train}) := \sum_{(u,i,r_{u,i}) \in D^{train}} \ell(r_{u,i}, \hat{r}_{u,i})$$

Mean Squared Error (MSE) is usually used as the loss function and RMSE for performance evaluation

Outline

MovieLens 100K

- Statistics
  - 100,000 ratings
  - 943 users
  - 1682 movies
- Ratings
  - 1(Worst) to 5(Best)
- File Format
  - *userid*|*itemid*|*rating*|*timestamp*.

Outline

Infrastructure and Coding Environment

1. Login to our server
   - Coding Environment : jupyter.ismll.de
   - User Name : vw1 to vw7
   - Password : Same as the user name
2. Create new python 3 notebook and rename it with your name
3. Import the following libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from collections import deque
import tensorflow as tf
from six import next
```

Outline

Data Preprocessing

- The dataset need to be loaded and splitted into train and test

```
def get_split_data():
    df_train = read_process("Data/ml100k/u1.base"
    , sep="\t")
    df_test = read_process("Data/ml100k/u1.test"
    , sep="\t")
    return df_train, df_test
```

- Read Function

```
def read_process(filname, sep="\t"):
    col_names = ["user", "item", "rate", "st"]
    df = pd.read_csv(filname, sep=sep, header=None, names=col_names, engine='python'
    df["user"] -= 1
    df["item"] -= 1
    for col in ("user", "item"):
        df[col] = df[col].astype(np.int32)
    df["rate"] = df["rate"].astype(np.float32)
    return df
```

Outline

6.1 User Average

- We use the average item rate as the prediction value for all test instances
- RMSE ?

6.1 User Average

- We use the average item rate as the prediction value for all test instances
- RMSE $= 1.062$

6.2 Item Average

- We use the average Item rate as the prediction value for all test instances
- RMSE ?

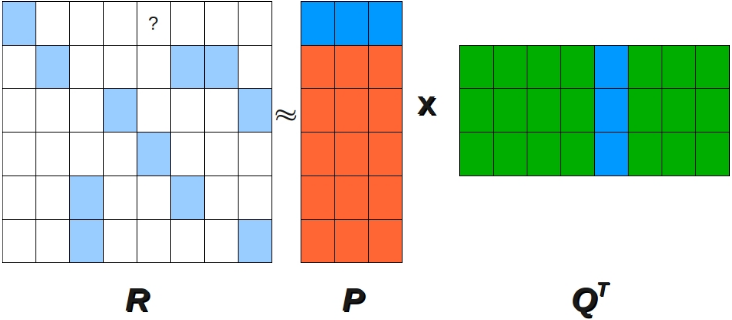
6.2 Item Average

- We use the average Item rate as the prediction value for all test instances
- RMSE = 1.090

6.3 - SVD (Factorization models)

- Each item $i \in I$ is associated with a latent feature vector $\mathbf{Q}_i \in \mathbb{R}^K$
- Each user $u \in U$ is associated with a latent feature vector $\mathbf{P}_u \in \mathbb{R}^K$
- Each entry in the original matrix can be estimated by

$$\hat{r}_{u,i} = \mathbf{P}_u^\top \mathbf{Q}_i = \sum_{k=1}^{K} P_{u,k} Q_{i,k}$$



6.3 - SVD (Objective Function)

Task:

$$\arg\min_{\mathbf{p},\mathbf{q}} \sum_{(u,i,r_{u,i})\in D^{train}} (r_{ui} - \hat{r}_{u,i})^2 + \lambda(||\mathbf{P}||^2 + ||\mathbf{Q}||^2)$$

Where:

- $\hat{r}_{u,i} := \mathbf{P}_u^\top \mathbf{Q}_i$
- D^{train} is the training data
- λ is a regularization constant

6.3 - SVD (SGD: gradients)

$$\mathcal{L} := \sum_{(u,i,r_{u,i})\in D^{train}} (r_{ui} - \hat{r}_{u,i})^2 + \lambda(||\mathbf{P}||^2 + ||\mathbf{Q}||^2) \tag{1}$$

$$\mathcal{L} := \sum_{(u,i,r_{u,i})\in D^{train}} \mathcal{L}_{u,i} \tag{2}$$

Gradients:

$$\frac{\partial \mathcal{L}_{u,i}}{\partial \mathbf{P}_{u,k}} = -2(r_{u,i} - \hat{r}_{u,i})\mathbf{Q}_{i,k} + 2\lambda \mathbf{P}_{u,k}$$

$$\frac{\partial \mathcal{L}_{u,i}}{\partial \mathbf{Q}_{i,k}} = -2(r_{u,i} - \hat{r}_{u,i})\mathbf{P}_{u,k} + 2\lambda \mathbf{Q}_{i,k}$$

6.3 - SVD (Stochastic Gradient Descent Algorithm)

procedure LEARNLATENTFACTORS
input: D^{Train}, λ, α
  $(\mathbf{p}_u)_{u \in U} \sim N(0, \sigma \mathbf{I})$
  $(\mathbf{q}_i)_{i \in I} \sim N(0, \sigma \mathbf{I})$
  repeat
    for $(u, i, r_{u,i}) \in D^{Train}$ **do** ▷ In a random order
      $\xi_{u,i} = r_{u,i} - \hat{r}_{u,i}$
      for $k = 1, \ldots, K$ **do**
        $\mathbf{P}_{u,k} \leftarrow \mathbf{P}_{u,k} + \alpha \left(\xi_{u,i} \mathbf{Q}_{i,k} - \lambda \mathbf{P}_{u,k} \right)$
        $\mathbf{Q}_{i,k} \leftarrow \mathbf{Q}_{i,k} + \alpha \left(\xi_{u,i} \mathbf{P}_{u,k} - \lambda \mathbf{Q}_{i,k} \right)$
      end for
    end for
  until convergence
  return $\mathbf{P}, \mathbf{Q}$
end procedure

6.3 - SVD Implementation (Batch Generation)

- Random batches for training

```
class ShuffleIterator(object):
    def __init__(self, inputs, batch_size=10):
        self.inputs = inputs
        self.batch_size = batch_size
        self.num_cols = len(self.inputs)
        self.len = len(self.inputs[0])
        self.inputs = np.transpose(np.vstack([np.array(self.inputs[i]) for i in rang
    def __len__(self):
        return self.len
    def __iter__(self):
        return self
    def __next__(self):
        return self.next()
    def next(self):
        ids = np.random.randint(0, self.len, (self.batch_size,))
        out = self.inputs[ids, :]
        return [out[:, i] for i in range(self.num_cols)]
```

6.3 - SVD Implementation (Batch Generation)

- Sequentially generated batches for test data

```
class OneEpochIterator(ShuffleIterator):
    def __init__(self, inputs, batch_size=10):
        super(OneEpochIterator, self).__init__(inputs, batch_size=batch_size)
        if batch_size > 0:
            self.idx_group = np.array_split(np.arange(self.len), np.ceil(self.len / b
        else:
            self.idx_group = [np.arange(self.len)]
        self.group_id = 0

    def next(self):
        if self.group_id >= len(self.idx_group):
            self.group_id = 0
            raise StopIteration
        out = self.inputs[self.idx_group[self.group_id], :]
        self.group_id += 1
        return [out[:, i] for i in range(self.num_cols)]
```

6.3 - SVD Implementation (Model architecture)

```
def inference_svd(user_batch, item_batch, user_num, item_num, dim=5, device="/cpu:0
    with tf.device("/cpu:0"):
        w_user = tf.get_variable("embd_user", shape=[user_num, dim],
            initializer=tf.truncated_normal_initializer(stddev=0.02))
        w_item = tf.get_variable("embd_item", shape=[item_num, dim],
            initializer=tf.truncated_normal_initializer(stddev=0.02))
        embd_user = tf.nn.embedding_lookup(w_user, user_batch)
        embd_item = tf.nn.embedding_lookup(w_item, item_batch)
        infer = tf.reduce_sum(tf.multiply(embd_user, embd_item), 1)
        regularizer = tf.add(tf.nn.l2_loss(embd_user)
        , tf.nn.l2_loss(embd_item), name="svd_regularizer")
    return infer, regularizer
```

6.3 - SVD Implementation (Optimization Function)

```
def optimization(infer, regularizer, rate_batch, learning_rate=0.001
, reg=0.0, device="/cpu:0"):
    global_step = tf.train.get_global_step()
    assert global_step is not None
    with tf.device(device):
        costl2 = tf.nn.l2_loss(tf.subtract(infer, rate_batch))
        penalty = tf.constant(reg, dtype=tf.float32, shape=[]
        , name="l2")
        cost = tf.add(costl2, tf.multiply(regularizer, penalty))
        train_op = tf.train.AdamOptimizer(learning_rate).minimize(cost
         ,global_step=global_step)
    return cost, train_op
```

6.3 - SVD Implementation (Main Function Part1)

```
def svd(train, test):
    samples_per_batch = len(train) // BATCH_SIZE

    iter_train = ShuffleIterator([train["user"],train["item"]
    ,train["rate"]],batch_size=BATCH_SIZE)
    iter_test = OneEpochIterator([test["user"],test["item"]
    ,test["rate"]],batch_size=-1)
    user_batch = tf.placeholder(tf.int32, shape=[None],name="id_user")
    item_batch = tf.placeholder(tf.int32, shape=[None],name="id_item")
    rate_batch = tf.placeholder(tf.float32, shape=[None])
    infer, regularizer = inference_svd(user_batch, item_batch
    , user_num=USER_NUM, item_num=ITEM_NUM, dim=DIM,
                                        device=DEVICE)
    global_step = tf.contrib.framework.get_or_create_global_step()
    _, train_op = optimization(infer, regularizer, rate_batch
    , learning_rate=0.001, reg=0.09, device=DEVICE)
```

6.3 - SVD Implementation (Main Function Part2)

```
init_op = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init_op)
    print("{}␣{}␣{}␣{}".format("epoch", "train_error", "val_error"
    , "elapsed_time"))
    errors = deque(maxlen=samples_per_batch)
    start = time.time()
    for i in range(EPOCH_MAX * samples_per_batch):
        users, items, rates = next(iter_train)
        _, pred_batch = sess.run([train_op, infer]
            , feed_dict={user_batch: users,item_batch: items
                ,rate_batch: rates})
        pred_batch = clip(pred_batch)
        errors.append(np.power(pred_batch - rates, 2))
        if i % samples_per_batch == 0:
            train_err = np.sqrt(np.mean(errors))
            test_err2 = np.array([])
            for users, items, rates in iter_test:
                pred_batch = sess.run(infer,
                    feed_dict={user_batch: users,item_batch: items})
                pred_batch = clip(pred_batch)
                test_err2 = np.append(test_err2
                , np.power(pred_batch - rates, 2))
```

6.3 - SVD Implementation (Main Function Part3)

```
end = time.time()
test_err = np.sqrt(np.mean(test_err2))
print("{:3d}␣{:f}␣{:f}␣{:f}(s)".format(i // samples_per_batch
, train_err, test_err,end - start))
start = end
```

6.3 - SVD Experiment

- What is the error value if we set the dimension to 50, batch size to 1000 and max epochs to 100 ? What happens after the 60 epoch?

6.3 - SVD Experiment

- What is the error value if we set the dimension to 50, batch size to 1000 and max epochs to 100 ? What happens after the epoch 60?
  - **Over-Fitting**
  - **RMSE = 0.932**
- How to solve it?

6.3 - SVD Experiment

- What is the error value if we set the dimension to 50, batch size to 1000 and max epochs to 100 ? What happens after the 60 epoch?
  Over-Fitting
- How to solve it?
  With Regularization Penalty = 0.09
  RMSE = 0.922

Thank You