628 Final Project Report
Event Recommendation System

Xinghan Qin

Abstract

This report will discuss the whole processing of why and how to build Event Recommendation System, system performance analyse and future improvement.

1 Introduction

As life speed becomes faster and faster, people sometimes can feel lonely and bored but don't know what to do. One of the reasons for this situation is because our world is too colourful. When people have too many interests, then sometimes it will become very hard for us to decide exactly what to do. And at the end, we will decide to do nothing.

Hence, because of this problem, I decided to create an events recommendation system to help people choose some events, that this person may interest to, based on the person information, friends list, and event attendance record. The data set was found on open source website, Kaggle. Following is the link:

`https://www.kaggle.com/c/event-recommendation-engine-challenge/data`

2 Aims and Objectives

2.1 Aims

Based on given events information, user information, training data and test data, to train a network to predict if the user will interest or uninterest to the event.

2.2 Objectives

• Analyse the given data sets and select the features
• Read and process data into wanted format
• Create the model based on requirement
• Train and evaluate the model by train data and validate data
• Predict the result for test data

3 Data Analyse

3.1 Data sets

• train.csv: (15398 * 6)
user, event, invited, timestamp, interested, not_interested
• test.csv: (10237 * 4)
user, event, invited, timestamp

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

• users.csv: (38209 * 7)
user_id, locale, birthyear, gender, joinedAt, location, timezone
• event_attendees.csv: (24144 * 5)
event, yes, maybe, invited, no
• events.csv: (3137972 * 110)
event_id, user_id, start time, city, state, zip, country, lat, lng, count_1, count_2, ..., count_100, count_other
• user_friends.csv: (38202 * 2)
user_id, friends

3.2 Features

• user_id: the id of the user
• event_id: the id of the event
• interested: does user interest in the event
• not_interested: does user uninterested the event
• invited: have user been invited, 1 for yes, 0 for no
• friend_attend_yes: the number of friends attend the event
• friend_attend_maybe: the number of friends maybe attend the event
• friend_attend_invited: the number of friends invited to the event
• friend_attend_no: the number of friends not attend the event
• attend_same_genderrate_yes: the rate of same gender attends the event
• attend_same_gender_rate_maybe: the rate of same gender maybe attends the event
• attend_same_gender_rate_invite: the rate of same gender invites by the event
• attend_same_gender_rate_no: the rate of same gender not attend the event
• host_is_friend: is host is friend of user, 1 for yes, 0 for no
• c_1 to c_100: the information about the event
• interest_c1 to interest_c_100: the mean value of user attended events' information

More features will be added in the future. The information about time, location and time zone has not been applied.

4 Data Processing

4.1 Read data

To read and use given data sets efficiently, 7 dictionaries were designed:
("1": yes, "2": maybe, "3": invited, "4": no)

• train_dict = {user_id : {event_id : [invited, interested, not_interested]}}
• test_dict = {user_id : {event_id : invited}}
• friends_dict = {user_id : []}
• users_dict = {user_id : [birthday, gender]}
• events_dict = {event_id : [host_user_id, [c_list]]}
• user_interests_dict = {user_id : {"1" : [], "2" : [], "3" : [], "4" : []}}
• event_attendees_dict = {event_id : {"1" : [], "2" : [], "3" : [], "4" : []}}

During reading the data, Duplicated data in user was found, and shows in Figure 1 and Figure 2. Because it was just duplicated and had no multiple data for same combo of user and event, the data will be ignored if it just duplicated. Hence, after processing, the train_data and test_data would be 15398 – 178 and 10237 – 131 and result shows in Figure 3.

```
In [98]:
train_dict = {}
for index, row in train.iterrows():
    user_id = str(row[0])
    event_id = str(row[1])
    invited = row[2]
    interested = row[4]
    not_interested = row[5]
    if not user_id in train_dict:
        train_dict[user_id] = {}
    if event_id in train_dict[user_id]:
        if train_dict[user_id][event_id] == [invited, interested, not_interested]:
            print("Duplicated data! for event_id, user_id: " + event_id + ", " + user_id)
        else:
            print("Multiple data! for event_id, user_id: " + event_id + ", " + user_id)
    else:
        train_dict[user_id][event_id] = [invited, interested, not_interested]

Duplicated data! for event_id, user_id:1462902079, 203456139
Duplicated data! for event_id, user_id:4242816413, 203456139
Duplicated data! for event_id, user_id:498238691, 203456139
Duplicated data! for event_id, user_id:745914541, 203456139
```

Figure 1: Duplicated data

```
In [148]:
train_dict = create_train_dict(train)

Duplicated data in user: 178
Multiple data in user: 0

In [149]:
test_dict = create_test_dict(test)

Duplicated data in user: 131
Multiple data in user: 0
```

Figure 2: Amount of d duplicated data

```
print(train.shape)
print(test.shape)

(15220, 214)
(10106, 214)
```

Figure 3: Size of training and testing data

4.2 Data processing

As all data has been read into the dictionary, train_data and test_data should be create based on the feature design.

• user_id:
train_dict.keys()
• event_id:
train_dict[user_id].keys()
• interested:
train_dict[user_id][event_id][1], for test_data is -1
• not_interested:
train_dict[user_id][event_id][2], for test_data is -1
• invited:
train_dict[user_id][event_id][0]
• friend_attend_yes, friend_attend_maybe, friend_attend_invited, friend_attend_no:
attend_list = event_attendees_dict[event_id]
friends_list = friends_dict[user_id]
compare attend_list with friends_list
• attend_same_gender_rate_yes, attend_same_gender_rate_maybe, attend_same_gender_rate_invite, attend_same_gender_rate_no:
attend_list = event_attendees_dict[event_id]
gender = users_dict[guest_id in attend_list]
• host_is_friend:

friends_list = friends_dict[user_id]
host_id = events_dict[event_id][0]
• c_1 to c_100:
events_dict[event_id][1 : 100]
• interest_c1 to interest_c_100:
interests_list = user_interests_dict[user_id]
mean of all events_dict[event_id in interests_list][1 : 100]

During the data processing, some problems about the data set were found. The main problem was lack of data. For example, lots of user_id in event_attendees cannot be found in users file. Hence, while processing the same gender, there were lots of unknown data. This situation also happened for user_interests_dict. Some user didn't attend any events before, hence, there was not record for the interest. Because lack of data happens in real world as well, unknown data was ignored or tread as 0 this time depends on the situation.

4.3 Result

train_data.csv will be showed in Figure 4. test_data.csv will be showed in Figure 5.

```
1 train
```

0	3044012	1918771225	0	0	0	0	0	3	3	0.000000	...	0.0
1	3044012	1502284248	0	0	0	0	0	1	0	0.000000	...	0.0
2	3044012	2529072432	1	0	0	0	0	2	0	0.833333	...	0.0
3	3044012	3072478280	0	0	0	1	0	0	0	0.000000	...	0.0
4	3044012	1390707377	0	0	0	0	0	2	0	0.333333	...	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
15215	4293103086	2750873665	0	0	0	0	1	0	0	0.000000	...	0.0
15216	4293103086	4084655790	0	0	0	1	0	1	0	0.000000	...	0.0
15217	4293103086	598708806	0	0	0	1	1	3	1	0.000000	...	0.0
15218	4293103086	604179853	0	0	0	0	1	0	1	0.000000	...	0.0
15219	4293103086	2351245308	1	0	0	0	0	1	0	0.000000	...	0.0

15220 rows × 214 columns

Figure 4: Train data

```
1 test
```

	user_id	event_id	interested	un_interested	invited	friend_yes	friend_maybe	friend_invited	friend_no	same_gender_yes	...	interest_c_190	inter
0	1776192	2877501688	-1	-1	0	1	0	1	0	0.0	...	0.0	
1	1776192	3025444328	-1	-1	0	1	0	11	2	0.0	...	0.0	
2	1776192	4078218285	-1	-1	0	0	0	7	0	0.0	...	0.0	
3	1776192	1024025121	-1	-1	0	0	0	3	0	0.0	...	0.0	
4	1776192	2972428928	-1	-1	0	0	0	3	0	0.0	...	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
10101	4289724715	2652356640	-1	-1	0	0	0	0	0	1.0	...	0.0	
10102	4289724715	3128167927	-1	-1	0	0	0	0	0	0.0	...	0.0	
10103	4289724715	645947926	-1	-1	0	0	0	0	0	0.0	...	0.0	
10104	4289724715	3547653054	-1	-1	0	0	0	0	0	0.0	...	0.0	
10105	4289724715	2769292935	-1	-1	0	0	0	0	0	0.0	...	0.0	

10106 rows × 214 columns

Figure 5: Test data

5 Model Training

5.1 Data prepare

Total training data set has 15220 elements, 14000 elements were set as training data and the rest data were set as validation data. Total test data set has 10106 elements, Figure 6.

```
In [5]:
train_total_size = 15220
training_size = 14000
validate_size = train_total_size - training_size
testing_size = 10106

training_data = torch.FloatTensor(train.iloc[: training_size, 4 :].values)
validate_data = torch.FloatTensor(train.iloc[training_size :, 4 :].values)
testing_data = torch.FloatTensor(test.iloc[:, 4 :].values)
training_label = torch.FloatTensor(train.iloc[: training_size, [2, 3]].values)
validate_label = torch.FloatTensor(train.iloc[training_size :, [2, 3]].values)
print("training_data size:", list(training_data.shape))
print("validate_data size:", list(validate_data.shape))
print("testing_data size:", list(testing_data.shape))
print("training_label size:", list(training_label.shape))
print("validate_label size:", list(validate_label.shape))

training_data size: [14000, 210]
validate_data size: [1220, 210]
testing_data size: [10106, 210]
training_label size: [14000, 2]
validate_label size: [1220, 2]
```

Figure 6: Size of all data sets

5.2 Model setting

First hidden layer: input size 210, output size 514, linear and ReLU
Second hidden layer: input size 514, output size 128, linear and ReLU
Output layer: input size 128, output size 2, linear

```
myNet(
  (fc1): Linear(in_features=210, out_features=514, bias=True)
  (relu1): ReLU()
  (fc2): Linear(in_features=514, out_features=128, bias=True)
  (relu2): ReLU()
  (fc3): Linear(in_features=128, out_features=2, bias=True)
)
```

Figure 7: Model

Optimizer and loss function in Figure 8

```
optimizer = optim.Adam(net.parameters(), lr=lr)
loss_funtion = nn.MSELoss()
```

Figure 8: Optimizer and loss function

Depends on the batch_size to create train_loader in Figure 9

```python
def load_array(data_arrays, batch_size, is_train=True):
    features, labels = data_arrays
    num_examples = len(labels)
    indices = list(range(num_examples))
    random.shuffle(indices)
    array = []
    for i in range(0, num_examples, batch_size):
        batch_indices = torch.tensor(indices[i : min(i + batch_size, num_examples - 1)])
        minibatch_features = features.index_select(0, batch_indices)
        minibatch_labels = labels.index_select(0, batch_indices)
        array.append((minibatch_features, minibatch_labels))
    train_loader = iter(array)
    return train_loader
```

Figure 9: Train loader

Depends on Epochs to train the model, and print the loss for this epoch in Figure 10

```python
for epoch in range(Epchs):
    data_iter = load_array((training_data, training_label), batch_size)
    for batch_X, batch_y in data_iter:
        net.zero_grad()
        output = net(batch_X)
        loss = loss_funtion(output, batch_y)
        loss.backward()
        optimizer.step()
    if epoch % ((Epchs - 1) / 10) == 0:
        print("loss for epoch ",epoch, " is ", loss.data)
```

Figure 10: Training circle

5.3 Parameters setting

To improve the model, batch_size, learning rate(lr) and Epchs were adjusted based on the final epoch loss and AUC of results.

For batch_size = 140, lr = 0.01, Epchs = 101:
As Figure 11 shows, the final epoch loss was 0.0546. And during the training, the loss didn't get improved gradually. Hence, lr was decided to be decreased.

```
loss for epoch  0  is  tensor(0.1297)
loss for epoch  10  is  tensor(0.0823)
loss for epoch  20  is  tensor(0.0780)
loss for epoch  30  is  tensor(0.0868)
loss for epoch  40  is  tensor(0.0742)
loss for epoch  50  is  tensor(0.0755)
loss for epoch  60  is  tensor(0.0734)
loss for epoch  70  is  tensor(0.0737)
loss for epoch  80  is  tensor(0.0643)
loss for epoch  90  is  tensor(0.0873)
loss for epoch  100  is  tensor(0.0676)
```

In [13]:
```python
train_result = net(training_data)
print(train_result.shape)
print("loss for train_data is: ", loss_funtion(train_result, training_label).data)
fpr, tpr, thresholds = metrics.roc_curve(training_label.data.numpy()[:, 0], train_result.data.numpy()[:, 0], pos_la
train_auc = metrics.auc(fpr, tpr)
print("AUC is: ", train_auc)
```

```
torch.Size([14000, 2])
loss for train_data is:  tensor(0.0546)
AUC is:  0.9252576997007824
```

In [14]:
```python
result = net(validate_data)
print(result.shape)
print("loss for train_data is: ", loss_funtion(result, validate_label).data)
fpr, tpr, thresholds = metrics.roc_curve(validate_label.data.numpy()[:, 0], result.data.numpy()[:, 0], pos_label=1)
validate_auc = metrics.auc(fpr, tpr)
print("AUC is: ", validate_auc)
```

```
torch.Size([1220, 2])
loss for train_data is:  tensor(0.1603)
AUC is:  0.5952877846790889
```

Figure 11: 140, 0.01, 101

For batch_size = 140, lr = 0.001, Epchs = 101:
In Figure 12, as lr was decreased, the loss has been decreased. However, because in the training data, there were lots of 0 (the reason will be discus in the next part). batch_size was increased to avoid all data in batch were 0.

```
loss for epoch  0  is  tensor(0.1149)
loss for epoch  10  is  tensor(0.0769)
loss for epoch  20  is  tensor(0.0520)
loss for epoch  30  is  tensor(0.0541)
loss for epoch  40  is  tensor(0.0497)
loss for epoch  50  is  tensor(0.0436)
loss for epoch  60  is  tensor(0.0425)
loss for epoch  70  is  tensor(0.0228)
loss for epoch  80  is  tensor(0.0327)
loss for epoch  90  is  tensor(0.0293)
loss for epoch  100  is  tensor(0.0371)
```

In [13]:
```
train_result = net(training_data)
print(train_result.shape)
print("loss for train_data is: ", loss_funtion(train_result, training_label).data)
fpr, tpr, thresholds = metrics.roc_curve(training_label.data.numpy()[:, 0], train_result.data.numpy()[:, 0], pos_la
train_auc = metrics.auc(fpr, tpr)
print("AUC is: ", train_auc)
```

```
torch.Size([14000, 2])
loss for train_data is:  tensor(0.0259)
AUC is:  0.985314843438633
```

In [14]:
```
result = net(validate_data)
print(result.shape)
print("loss for train_data is: ", loss_funtion(result, validate_label).data)
fpr, tpr, thresholds = metrics.roc_curve(validate_label.data.numpy()[:, 0], result.data.numpy()[:, 0], pos_label=1)
validate_auc = metrics.auc(fpr, tpr)
print("AUC is: ", validate_auc)
```

```
torch.Size([1220, 2])
loss for train_data is:  tensor(0.1678)
AUC is:  0.571976811594203
```

Figure 12: 140, 0.001, 101

For batch_size = 1400, lr = 0.001, Epchs = 101:
As Figure 13 shows, the loss increased. However, different than before, the loss kept tend of decrease. The previous results tended to be stable or jump back. To determine when would the results tended to be stable or jump back, Epchs was increased.

```
loss for epoch  0  is  tensor(0.1319)
loss for epoch  10  is  tensor(0.1486)
loss for epoch  20  is  tensor(0.0961)
loss for epoch  30  is  tensor(0.0824)
loss for epoch  40  is  tensor(0.0630)
loss for epoch  50  is  tensor(0.0580)
loss for epoch  60  is  tensor(0.0535)
loss for epoch  70  is  tensor(0.0541)
loss for epoch  80  is  tensor(0.0468)
loss for epoch  90  is  tensor(0.0478)
loss for epoch  100  is  tensor(0.0394)
```

In [13]:
```
train_result = net(training_data)
print(train_result.shape)
print("loss for train_data is: ", loss_funtion(train_result, training_label).data)
fpr, tpr, thresholds = metrics.roc_curve(training_label.data.numpy()[:, 0], train_result.data.numpy()[:, 0], pos_la
train_auc = metrics.auc(fpr, tpr)
print("AUC is: ", train_auc)
```

```
torch.Size([14000, 2])
loss for train_data is:  tensor(0.0404)
AUC is:  0.9617783759299537
```

In [14]:
```
result = net(validate_data)
print(result.shape)
print("loss for train_data is: ", loss_funtion(result, validate_label).data)
fpr, tpr, thresholds = metrics.roc_curve(validate_label.data.numpy()[:, 0], result.data.numpy()[:, 0], pos_label=1)
validate_auc = metrics.auc(fpr, tpr)
print("AUC is: ", validate_auc)
```

```
torch.Size([1220, 2])
loss for train_data is:  tensor(0.1630)
AUC is:  0.6069747412008282
```

Figure 13: 1400, 0.001, 101

For batch_size = 1400, lr = 0.001, Epchs = 1001:
As Figure 14 shows, the loss was decreased. However, although the AUC of training_data_pred became 0.994, which means the model is very close to the training data, the AUC of validation data decreased to 0.538. This caused by overfitting. Because the model has been trained too many times, the model almost kept the shape of training data. Hence, Epchs needed to be decreased. As the part be highlighted shows, during 700th epoch, the loss has jumped back. Hence, Epchs was set somewhere around 600.

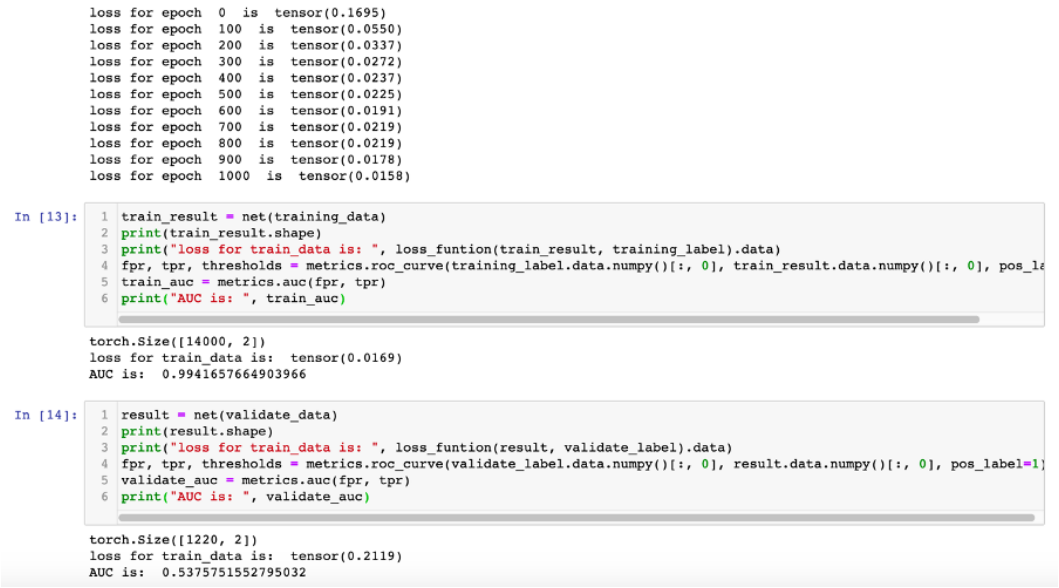

Figure 14: 1400, 0.001, 1001

For batch_size = 1400, lr = 0.001, Epchs = 651:
As Figure 15 shows, the loss was 0.0184, and AUC of validation data was 0.581. Hence, the model has been improved, and this would be the final version of the model.

6 Result Analyse

As the final result shows, the final AUC of validation data was 0.581 and loss was 0.2, which was one of the best results and parameter setting for this model. However, this result did not match the expectation. Following are the reasons may cause this situation:

```
loss for epoch  0  is  tensor(0.3842)
loss for epoch  65  is  tensor(0.0503)
loss for epoch  130  is  tensor(0.0392)
loss for epoch  195  is  tensor(0.0341)
loss for epoch  260  is  tensor(0.0251)
loss for epoch  325  is  tensor(0.0235)
loss for epoch  390  is  tensor(0.0263)
loss for epoch  455  is  tensor(0.0191)
loss for epoch  520  is  tensor(0.0219)
loss for epoch  585  is  tensor(0.0235)
loss for epoch  650  is  tensor(0.0216)
```

In [13]:

```
train_result = net(training_data)
print(train_result.shape)
print("loss for train_data is: ", loss_funtion(train_result, training_label).data)
fpr, tpr, thresholds = metrics.roc_curve(training_label.data.numpy()[:, 0], train_result.data.numpy()[:, 0], pos_la
train_auc = metrics.auc(fpr, tpr)
print("AUC is: ", train_auc)
```

```
torch.Size([14000, 2])
loss for train_data is:  tensor(0.0184)
AUC is:  0.9931112853710314
```

In [14]:

```
result = net(validate_data)
print(result.shape)
print("loss for train_data is: ", loss_funtion(result, validate_label).data)
fpr, tpr, thresholds = metrics.roc_curve(validate_label.data.numpy()[:, 0], result.data.numpy()[:, 0], pos_label=1)
validate_auc = metrics.auc(fpr, tpr)
print("AUC is: ", validate_auc)
```

```
torch.Size([1220, 2])
loss for train_data is:  tensor(0.1993)
AUC is:  0.5813813664596273
```

Figure 15: 1400, 0.001, 651

• Lack of data:
While processing the training_data and testing_data based the features been designed, there were lots of unknow data. For lots of user_id in friends_list, no information can be found according to user_id in users.csv. This means lots of the numbers of friend's attendance for the user were 0. Also, for user_id in attended.csv had same issue. For the user in training data, lots of them did not have any record of the event attendance. Hence, the result for interest_c_1 to interest_c_100 could all be 0. The lack of data caused training data included many 0.

• Unused data:
In the original data set, there were some data this model didn't apply. Those data were mainly location, timestamp, birthday and time zone. For those data, some needed higher skill to processing. Like location, the given data in same column has different format. Some were city and status, and some others were status and country. Hence, need other tool to determine the information. And for data like timestamp, with limited knowledge, those data cannot be transfer into useful data. Put timestamp in integer, like 20200314, into training data was not meaningful. Hence, some of unused data may be important features for model. However, based on limited skill and knowledge, those data have not been applied.

• Model:
The model can be more complicated by applying different type of method.

• Parameters:
More testing should be done to improve the parameters.

7 Future Improvement

As been discussed in Result analyse, following are the future improvement:

• Search for more data set to complete the training data (the data was found on Kaggle, hence this is not doable)
• Apply more data in the original data set, learn more skill and theory to represent these data
• Complicate the model by using more methods
• Do more testing to improve the parameters

8 Conclusion

For this event recommendation system model, it has 214 features, the loss of training data was 0.0184, the AUC of training data was 0.993, the loss for validation data was 0.2 and AUC of validation data was 0.831. Although the trained model did not have high performers, the improve directions have been found out. With more self-learning, it will have more useful features, more data, more complicated model and better parameters.