In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import keras
import keras.layers
import tensorflow as tf
train_data = pd.read_csv("gs://zw2624-bucket/input/large_train.csv")
test_data = pd.read_csv("gs://zw2624-bucket/input/large_test.csv")
train_data = train_data.rename(columns={"userId": "user_id", "movieId": "business_id"})
test_data = test_data.rename(columns={"userId": "user_id", "movieId": "business_id"})
train_data['is_train'] = True
test_data['is_train'] = False
all_data = pd.concat([train_data,test_data])
all_data.user_id = all_data.user_id.astype('category').cat.codes.values
all_data.business_id = all_data.business_id.astype('category').cat.codes.values
is_train = all_data['is_train'] == True
train_data = all_data[is_train]
test_data = all_data[~is_train]
n_latent_factors_user = 8
n_latent_factors_business = 10
n_latent_factors_mf = 3
n_users, n_business = len(all_data.user_id.unique()), len(all_data.business_id.unique())
```

```
Using TensorFlow backend.
```

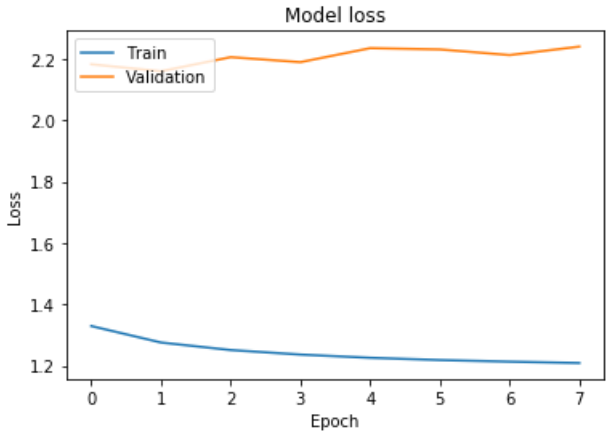
Build Model

In [4]:
```
business_input = keras.layers.Input(shape=[1],name='Item')
business_embedding_mlp = keras.layers.Embedding(n_business + 1, n_latent_factors_business, name
='Business-Embedding-MLP')(business_input)
business_vec_mlp = keras.layers.Flatten(name='FlattenBusiness-MLP')(business_embedding_mlp)
business_vec_mlp = keras.layers.Dropout(0.2)(business_vec_mlp)

business_embedding_mf = keras.layers.Embedding(n_business + 1, n_latent_factors_mf, name='busin
ess-Embedding-MF')(business_input)
business_vec_mf = keras.layers.Flatten(name='Flattenbusiness-MF')(business_embedding_mf)
business_vec_mf = keras.layers.Dropout(0.2)(business_vec_mf)

user_input = keras.layers.Input(shape=[1],name='User')
user_vec_mlp = keras.layers.Flatten(name='FlattenUsers-MLP')(keras.layers.Embedding(n_users + 1
, n_latent_factors_user,name='User-Embedding-MLP')(user_input))
user_vec_mlp = keras.layers.Dropout(0.2)(user_vec_mlp)

user_vec_mf = keras.layers.Flatten(name='FlattenUsers-MF')(keras.layers.Embedding(n_users + 1,
n_latent_factors_mf,name='User-Embedding-MF')(user_input))
user_vec_mf = keras.layers.Dropout(0.2)(user_vec_mf)

concat = keras.layers.concatenate([business_vec_mlp, user_vec_mlp], name='Concat')

concat_dropout = keras.layers.Dropout(0.2)(concat)
dense = keras.layers.Dense(128, activation='relu',name='FullyConnected')(concat_dropout)
dense_batch = keras.layers.BatchNormalization(name='Batch')(dense)
dropout_1 = keras.layers.Dropout(0.2,name='Dropout-1')(dense_batch)
dense_2 = keras.layers.Dense(64,name='FullyConnected-1')(dropout_1)
dense_batch_2 = keras.layers.BatchNormalization(name='Batch-2')(dense_2)

dropout_2 = keras.layers.Dropout(0.2,name='Dropout-2')(dense_batch_2)
dense_3 = keras.layers.Dense(50,name='FullyConnected-2')(dropout_2)
dense_4 = keras.layers.Dense(20,name='FullyConnected-3', activation='relu')(dense_3)

pred_mf = keras.layers.concatenate([business_vec_mf, user_vec_mf], name='Dot')
pred_mlp = keras.layers.Dense(1, activation='relu',name='Activation')(dense_4)

combine_mlp_mf = keras.layers.concatenate([pred_mf,pred_mlp],name='Concat-MF-MLP')
result_combine = keras.layers.Dense(100,name='Combine-MF-MLP')(combine_mlp_mf)
deep_combine = keras.layers.Dense(100,name='FullyConnected-4')(result_combine)
result = keras.layers.Dense(1,name='Prediction')(deep_combine)
model = keras.Model([user_input, business_input], result)
model.compile(optimizer='adam',loss= 'mse', metrics =["accuracy", "mse"])
```

In [ ]:
```
history = model.fit([train_data.user_id.values, train_data.business_id.values], train_data.rati
ng_review,
                    epochs=8, validation_split=0.2,use_multiprocessing = True)
```

```
Train on 2894480 samples, validate on 723620 samples
Epoch 1/8
2894480/2894480 [==============================] - 3586s 1ms/step - loss: 1.3292 - accuracy:
0.3336 - mse: 1.3292 - val_loss: 2.1831 - val_accuracy: 0.1596 - val_mse: 2.1831
Epoch 2/8
2894480/2894480 [==============================] - 3603s 1ms/step - loss: 1.2755 - accuracy:
0.3486 - mse: 1.2755 - val_loss: 2.1605 - val_accuracy: 0.1657 - val_mse: 2.1605
Epoch 3/8
2894480/2894480 [==============================] - 3548s 1ms/step - loss: 1.2131 - accuracy:
0.3696 - mse: 1.2130 - val_loss: 2.2132 - val_accuracy: 0.1565 - val_mse: 2.2132
Epoch 8/8
2075776/2894480 [====================>.........] - ETA: 16:31 - loss: 1.2049 - accuracy: 0.37
31 - mse: 1.2049
```

In [110]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



Evaluation

all reviews

In [11]:
```
from sklearn.metrics import mean_squared_error
prediction = model.predict([test_data.user_id.values, test_data.business_id.values])
```

In [12]:
```
y_hat_rounded = np.round(prediction, 0)
print(np.sqrt(mean_squared_error(test_data.rating_review, y_hat_rounded)))
print(np.sqrt(mean_squared_error(test_data.rating_review, prediction)))
```

```
1.4239968457639733
1.3909788079504284
```

last review of each user

In [112]:
```
idx = test_data.groupby(['user_id'])['date_review'].transform(max) == test_data['date_review']
test_data_latest = test_data[idx]
print(np.sqrt(mean_squared_error(test_data_latest.rating_review, test_data_latest.predicted)))
```

```
1.428152827115814
```

user coverage (ranking)

In [111]:
```
test_data['predicted'] = pd.DataFrame(prediction)
correct = 0
for user in test_data.user_id.unique():
    real = np.argsort(test_data.loc[test_data['user_id'] == user]['rating_review'])
    pred = np.argsort(test_data.loc[test_data['user_id'] == user]['predicted'])
    correct += (list(real) == list(pred))
print(correct / len(test_data.user_id.unique()))
```

```
0.21452917029481278
```

In [31]:
```
pd.DataFrame(prediction).to_csv('gs://zw2624-bucket/output/large_data_relu_pred.csv')
```

In [88]:
```
popular_user_ID = pd.read_csv("gs://zw2624-bucket/input/yelp/popular_user_ID.csv")
popular_business_ID = pd.read_csv("gs://zw2624-bucket/input/yelp/popular_business_ID.csv")
midpopular_user_ID = pd.read_csv("gs://zw2624-bucket/input/yelp/midpopular_user_ID.csv")
midpopular_business_ID = pd.read_csv("gs://zw2624-bucket/input/yelp/midpopular_business_ID.csv"
)
unpopular_user_ID = pd.read_csv("gs://zw2624-bucket/input/yelp/unpopular_user_ID.csv")
unpopular_business_ID = pd.read_csv("gs://zw2624-bucket/input/yelp/unpopular_business_ID.csv")
```

In [66]:
```
train_data_2 = pd.read_csv("gs://zw2624-bucket/input/large_train.csv")
test_data_2 = pd.read_csv("gs://zw2624-bucket/input/large_test.csv")
train_data_2 = train_data_2.rename(columns={"userId": "user_id", "movieId": "business_id"})
test_data_2 = test_data_2.rename(columns={"userId": "user_id", "movieId": "business_id"})
train_data_2['is_train'] = True
test_data_2['is_train'] = False
all_data_2 = pd.concat([train_data_2,test_data_2])
all_data_2['user_id_code'] = all_data_2.user_id.astype('category').cat.codes.values
all_data_2['business_id_code'] = all_data_2.business_id.astype('category').cat.codes.values
is_train_2 = all_data_2['is_train'] == True
train_data_2 = all_data_2[is_train_2]
test_data_2 = all_data_2[~is_train_2]
```

Rmse for different user segments

In [68]:
```
test_data_pop = test_data_2.loc[test_data_2.user_id.isin(popular_user_ID.userId)]
test_data_mid = test_data_2.loc[test_data_2.user_id.isin(midpopular_user_ID.userId)]
test_data_unp = test_data_2.loc[test_data_2.user_id.isin(unpopular_user_ID.userId)]
```

In [71]:
```
prediction_pop = model.predict([test_data_pop.user_id_code.values,
                                                test_data_pop.business_id_code.values])
test_data_pop['predicted'] = pd.DataFrame(prediction_pop)
print(np.sqrt(mean_squared_error(test_data_pop.rating_review, prediction_pop)))

prediction_mid = model.predict([test_data_mid.user_id_code.values,
                                                test_data_mid.business_id_code.values])
test_data_mid['predicted'] = pd.DataFrame(prediction_mid)
print(np.sqrt(mean_squared_error(test_data_mid.rating_review, prediction_mid)))

prediction_unp = model.predict([test_data_unp.user_id_code.values,
                                                test_data_unp.business_id_code.values])
test_data_unp['predicted'] = pd.DataFrame(prediction_unp)
print(np.sqrt(mean_squared_error(test_data_unp.rating_review, prediction_unp)))
```

```
/opt/conda/anaconda/lib/python3.7/site-packages/ipykernel_launcher.py:3: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/
indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until

1.274947206881366

/opt/conda/anaconda/lib/python3.7/site-packages/ipykernel_launcher.py:8: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/
indexing.html#returning-a-view-versus-a-copy


1.3681039215085238
1.448215705477811

/opt/conda/anaconda/lib/python3.7/site-packages/ipykernel_launcher.py:13: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/
indexing.html#returning-a-view-versus-a-copy
  del sys.path[0]
```

Rmse for different business segments

In [89]:
```
test_data_pop_bus = test_data_2.loc[test_data_2.business_id.isin(popular_business_ID.businessId
)]
test_data_mid_bus = test_data_2.loc[test_data_2.business_id.isin(midpopular_business_ID.busines
sId)]
test_data_unp_bus = test_data_2.loc[test_data_2.business_id.isin(unpopular_business_ID.business
Id)]

prediction_pop_bus = model.predict([test_data_pop_bus.user_id_code.values,
                                    test_data_pop_bus.business_id_code.values])
test_data_pop_bus['predicted'] = pd.DataFrame(prediction_pop_bus)
print(np.sqrt(mean_squared_error(test_data_pop_bus.rating_review, prediction_pop_bus)))

prediction_mid_bus = model.predict([test_data_mid_bus.user_id_code.values,
                                    test_data_mid_bus.business_id_code.values])
test_data_mid_bus['predicted'] = pd.DataFrame(prediction_mid_bus)
print(np.sqrt(mean_squared_error(test_data_mid_bus.rating_review, prediction_mid_bus)))

prediction_unp_bus = model.predict([test_data_unp_bus.user_id_code.values,
                                    test_data_unp_bus.business_id_code.values])
test_data_unp_bus['predicted'] = pd.DataFrame(prediction_unp_bus)
print(np.sqrt(mean_squared_error(test_data_unp_bus.rating_review, prediction_unp_bus)))
```

```
/opt/conda/anaconda/lib/python3.7/site-packages/ipykernel_launcher.py:7: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/
indexing.html#returning-a-view-versus-a-copy
  import sys

1.2962228644750784

/opt/conda/anaconda/lib/python3.7/site-packages/ipykernel_launcher.py:12: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/
indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':

1.5672741297988584
1.6863254914941213

/opt/conda/anaconda/lib/python3.7/site-packages/ipykernel_launcher.py:17: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/
indexing.html#returning-a-view-versus-a-copy
```

Coverage for different user segments

In [ ]:
```
coverage_mid = 0
for user in test_data_mid.user_id_code.unique():
    real = np.argsort(test_data.loc[test_data['user_id'] == user]['rating_review'])
    pred = np.argsort(test_data.loc[test_data['user_id'] == user]['predicted'])
    coverage_mid += (list(real) == list(pred))
coverage_mid = coverage_mid / len(test_data_mid.user_id.unique())
print(coverage_mid)
```

```
0.20946297070864545
```

In [ ]:
```
coverage_unp = 0
for user in test_data_unp.user_id_code.unique():
    real = np.argsort(test_data.loc[test_data['user_id'] == user]['rating_review'])
    pred = np.argsort(test_data.loc[test_data['user_id'] == user]['predicted'])
    coverage_unp += (list(real) == list(pred))
coverage_unp = coverage_unp / len(test_data_unp.user_id.unique())
```

In [113]:
```
print(coverage_unp)
```

```
0.21668795232013624
```

In [ ]:
```
coverage_pop = 0
for user in test_data_pop.user_id_code.unique():
    real = np.argsort(test_data.loc[test_data['user_id'] == user]['rating_review'])
    pred = np.argsort(test_data.loc[test_data['user_id'] == user]['predicted'])
    coverage_pop += (list(real) == list(pred))
```

In [109]:
```
print(coverage_pop /len(test_data_pop.user_id.unique()))
```

```
0.21605637134794436
```

In [ ]: