In [40]:
```
import pandas as pd
import numpy as np
from zipfile import ZipFile
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from pathlib import Path
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve,roc_auc_score
```

In [79]:
```
movielens_dir = "../ml-25m/"
ratings_file = movielens_dir + "ratings.csv"
df = pd.read_csv(ratings_file)
```

In [3]:
```
df
```

Out[3]:

	userId	movieId	rating	timestamp
0	1	296	5.0	1147880044
1	1	306	3.5	1147868817
2	1	307	5.0	1147868828
3	1	665	5.0	1147878820
4	1	899	3.5	1147868510
...	...	...	...	...
25000090	162541	50872	4.5	1240953372
25000091	162541	55768	2.5	1240951998
25000092	162541	56176	2.0	1240950697
25000093	162541	58559	4.0	1240953434
25000094	162541	63876	5.0	1240952515

25000095 rows × 4 columns

In [67]:
```
df = df.sample(n = 1000000)
user_ids = df["userId"].unique().tolist()
user2user_encoded = {x: i for i, x in enumerate(user_ids)}
userencoded2user = {i: x for i, x in enumerate(user_ids)}
movie_ids = df["movieId"].unique().tolist()
movie2movie_encoded = {x: i for i, x in enumerate(movie_ids)}
movie_encoded2movie = {i: x for i, x in enumerate(movie_ids)}
df["user"] = df["userId"].map(user2user_encoded)
df["movie"] = df["movieId"].map(movie2movie_encoded)

num_users = len(user2user_encoded)
num_movies = len(movie_encoded2movie)
df["rating"] = df["rating"].values.astype(np.float32)
# min and max ratings will be used to normalize the ratings later
min_rating = min(df["rating"])
max_rating = max(df["rating"])
```

```
print(
    "Number of users: {}, Number of Movies: {}, Min rating: {}, Max rating: {}".form
        num_users, num_movies, min_rating, max_rating
    )
)
```

```
Number of users: 142453, Number of Movies: 23219, Min rating: 0.5, Max rating: 5.0
```

In [68]:
```
df
```

Out[68]:

	userId	movieId	rating	timestamp	user	movie
5301487	34392	1042	5.0	969930722	0	0
6369968	41327	287	5.0	835414251	1	1
7856699	51040	420	3.0	832239479	2	2
14253811	92287	2949	4.0	955083214	3	3
14259096	92328	4262	4.0	1447408206	4	4
...	...	...	...	...	...	...
14982269	97074	52606	2.0	1547076265	5148	9620
24301262	157937	3252	4.0	1456776868	8716	237
21465149	139523	194	4.0	1476945850	8493	3474
11829171	76682	185989	1.0	1525209365	11842	16104
11634914	75448	1407	4.0	992500633	72097	189

1000000 rows × 6 columns

In [69]:
```
df = df.sample(frac=1, random_state=42)
x = df[["user", "movie"]].values
# Normalize the targets between 0 and 1. Makes it easy to train.
y = df["rating"].apply(lambda x: (x - min_rating) / (max_rating - min_rating)).value
# Assuming training on 90% of the data and validating on 10%.
train_indices = int(0.9 * df.shape[0])
x_train, x_val, y_train, y_val = (
    x[:train_indices],
    x[train_indices:],
    y[:train_indices],
    y[train_indices:],
)
print(len(x_train))
print(len(x_val))
```

```
900000
100000
```

In [70]:
```
EMBEDDING_SIZE = 50


class RecommenderNet(keras.Model):
    def __init__(self, num_users, num_movies, embedding_size, **kwargs):
        super(RecommenderNet, self).__init__(**kwargs)
        self.num_users = num_users
        self.num_movies = num_movies
        self.embedding_size = embedding_size
        self.user_embedding = layers.Embedding(
```

```
            num_users,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6),
        )
        self.user_bias = layers.Embedding(num_users, 1)
        self.movie_embedding = layers.Embedding(
            num_movies,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6),
        )
        self.movie_bias = layers.Embedding(num_movies, 1)

    def call(self, inputs):
        user_vector = self.user_embedding(inputs[:, 0])
        user_bias = self.user_bias(inputs[:, 0])
        movie_vector = self.movie_embedding(inputs[:, 1])
        movie_bias = self.movie_bias(inputs[:, 1])
        dot_user_movie = tf.tensordot(user_vector, movie_vector, 2)
        # Add all the components (including bias)
        x = dot_user_movie + user_bias + movie_bias
        # The sigmoid activation forces the rating to between 0 and 1
        return tf.nn.sigmoid(x)


model = RecommenderNet(num_users, num_movies, EMBEDDING_SIZE)
model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(), optimizer=keras.optimizers.Adam(learn
)
```

In [76]:

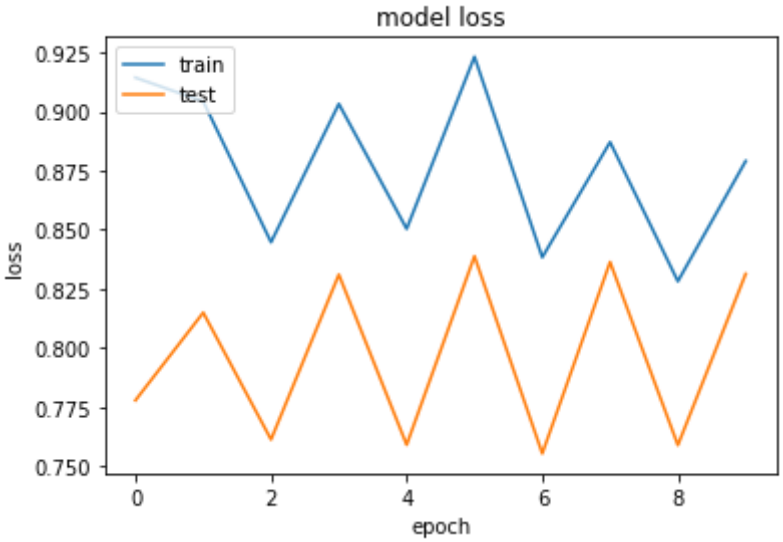
```
history = model.fit(
    x=x_train,
    y=y_train,
    batch_size=64,
    epochs=10,
    verbose=1,
    #steps_per_epoch=10000,
    validation_data=(x_val, y_val)
)
```

```
Epoch 1/10
14063/14063 [==============================] - 892s 63ms/step - loss: 0.9144 - val_l
oss: 0.7778
Epoch 2/10
14063/14063 [==============================] - 865s 62ms/step - loss: 0.9043 - val_l
oss: 0.8149
Epoch 3/10
14063/14063 [==============================] - 869s 62ms/step - loss: 0.8447 - val_l
oss: 0.7612
Epoch 4/10
14063/14063 [==============================] - 8639s 614ms/step - loss: 0.9032 - val
_loss: 0.8310
Epoch 5/10
14063/14063 [==============================] - 881s 63ms/step - loss: 0.8504 - val_l
oss: 0.7589
Epoch 6/10
14063/14063 [==============================] - 872s 62ms/step - loss: 0.9232 - val_l
oss: 0.8388
Epoch 7/10
14063/14063 [==============================] - 868s 62ms/step - loss: 0.8383 - val_l
oss: 0.7553
Epoch 8/10
```

```
14063/14063 [==============================] - 870s 62ms/step - loss: 0.8871 - val_l
oss: 0.8363
Epoch 9/10
14063/14063 [==============================] - 870s 62ms/step - loss: 0.8282 - val_l
oss: 0.7588
Epoch 10/10
14063/14063 [==============================] - 871s 62ms/step - loss: 0.8790 - val_l
oss: 0.8313
```

In [77]:

```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epoch")
plt.legend(["train", "test"], loc="upper left")
plt.show()
```



In [90]:

```
movie_df = pd.read_csv(movielens_dir + "movies.csv")

# Let us get a user and see the top recommendations.
user_id = df.userId.sample(1).iloc[0]
movies_watched_by_user = df[df.userId == user_id]
movies_not_watched = movie_df[
    ~movie_df["movieId"].isin(movies_watched_by_user.movieId.values)
]["movieId"]
movies_not_watched = list(
    set(movies_not_watched).intersection(set(movie2movie_encoded.keys()))
)
movies_not_watched = [[movie2movie_encoded.get(x)] for x in movies_not_watched]
user_encoder = user2user_encoded.get(user_id)
user_movie_array = np.hstack(
    ([[user_encoder]] * len(movies_not_watched), movies_not_watched)
)
ratings = model.predict(user_movie_array).flatten()
top_ratings_indices = ratings.argsort()[-10:][::-1]
recommended_movie_ids = [
    movie_encoded2movie.get(movies_not_watched[x][0]) for x in top_ratings_indices
]

print("Mostrando las recomendaciones para el usuario: {}".format(user_id))
print("====" * 9)
print("Peliculas calificadas por el usuario")
print("----" * 8)
top_movies_user = (
    movies_watched_by_user.sort_values(by="rating", ascending=False)
```

```
    .head(5)
    .movieId.values
)
movie_df_rows = movie_df[movie_df["movieId"].isin(top_movies_user)]
for row in movie_df_rows.itertuples():
    print(row.title, ":", row.genres)

print("----" * 8)
print("Top 10 peliculas recomendadas")
print("----" * 8)
recommended_movies = movie_df[movie_df["movieId"].isin(recommended_movie_ids)]
for row in recommended_movies.itertuples():
    print(row.title, ":", row.genres)
```

```
Mostrando las recomendaciones para el usuario: 74117
==================================
Peliculas calificadas por el usuario
--------------------------------
Mr. Holland's Opus (1995) : Drama
Legends of the Fall (1994) : Drama|Romance|War|Western
Shawshank Redemption, The (1994) : Crime|Drama
Forrest Gump (1994) : Comedy|Drama|Romance|War
Much Ado About Nothing (1993) : Comedy|Romance
--------------------------------
Top 10 peliculas recomendadas
--------------------------------
One Flew Over the Cuckoo's Nest (1975) : Drama
12 Angry Men (1957) : Drama
Ran (1985) : Drama|War
Godfather: Part II, The (1974) : Crime|Drama
Treasure of the Sierra Madre, The (1948) : Action|Adventure|Drama|Western
Great Escape, The (1963) : Action|Adventure|Drama|War
Lock, Stock & Two Smoking Barrels (1998) : Comedy|Crime|Thriller
Brief Encounter (1946) : Drama|Romance
Lady Eve, The (1941) : Comedy|Romance
Tokyo Story (Tôkyô monogatari) (1953) : Drama
```

In [106…

```
# Guardar el Modelo

model.save_weights("ModeloPrimeraPrueba", overwrite=True, save_format=None, options=
```

In [108…

```
# Cargar Modelo
model.load_weights("ModeloPrimeraPrueba", by_name=False, skip_mismatch=False, option
model.summary()
```

```
Model: "recommender_net_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_28 (Embedding)     multiple                  7122650
_________________________________________________________________
embedding_29 (Embedding)     multiple                  142453
_________________________________________________________________
embedding_30 (Embedding)     multiple                  1160950
_________________________________________________________________
embedding_31 (Embedding)     multiple                  23219
=================================================================
Total params: 8,449,272
Trainable params: 8,449,272
Non-trainable params: 0
_________________________________________________________________
```

In [109…

```
movie_df = pd.read_csv(movielens_dir + "movies.csv")

# Let us get a user and see the top recommendations.
user_id = df.userId.sample(1).iloc[0]
movies_watched_by_user = df[df.userId == user_id]
movies_not_watched = movie_df[
    ~movie_df["movieId"].isin(movies_watched_by_user.movieId.values)
]["movieId"]
movies_not_watched = list(
    set(movies_not_watched).intersection(set(movie2movie_encoded.keys()))
)
movies_not_watched = [[movie2movie_encoded.get(x)] for x in movies_not_watched]
user_encoder = user2user_encoded.get(user_id)
user_movie_array = np.hstack(
    ([[user_encoder]] * len(movies_not_watched), movies_not_watched)
)
ratings = model.predict(user_movie_array).flatten()
top_ratings_indices = ratings.argsort()[-10:][::-1]
recommended_movie_ids = [
    movie_encoded2movie.get(movies_not_watched[x][0]) for x in top_ratings_indices
]

print("Mostrando las recomendaciones para el usuario: {}".format(user_id))
print("====" * 9)
print("Peliculas calificadas por el usuario")
print("----" * 8)
top_movies_user = (
    movies_watched_by_user.sort_values(by="rating", ascending=False)
    .head(5)
    .movieId.values
)
movie_df_rows = movie_df[movie_df["movieId"].isin(top_movies_user)]
for row in movie_df_rows.itertuples():
    print(row.title, ":", row.genres)

print("----" * 8)
print("Top 10 peliculas recomendadas")
print("----" * 8)
recommended_movies = movie_df[movie_df["movieId"].isin(recommended_movie_ids)]
for row in recommended_movies.itertuples():
    print(row.title, ":", row.genres)
```

```
Mostrando las recomendaciones para el usuario: 61935
====================================
Peliculas calificadas por el usuario
--------------------------------
Toy Story (1995) : Adventure|Animation|Children|Comedy|Fantasy
Doors, The (1991) : Drama
Princess Bride, The (1987) : Action|Adventure|Comedy|Fantasy|Romance
Catch Me If You Can (2002) : Crime|Drama
Gods and Generals (2003) : Action|Drama|War
--------------------------------
Top 10 peliculas recomendadas
--------------------------------
Persuasion (1995) : Drama|Romance
Princess Mononoke (Mononoke-hime) (1997) : Action|Adventure|Animation|Drama|Fantasy
Dog Day Afternoon (1975) : Crime|Drama
Coming to America (1988) : Comedy|Romance
Frantic (1988) : Crime|Mystery|Thriller
Gorillas in the Mist (1988) : Drama
Imagine: John Lennon (1988) : Documentary
Land Before Time, The (1988) : Adventure|Animation|Children|Fantasy
Wild Strawberries (Smultronstället) (1957) : Drama
Grave of the Fireflies (Hotaru no haka) (1988) : Animation|Drama|War
```

In [ ]: