

Recommender Systems

Spyros Samothrakis
Lecturer/Assistant Director@IADS
University of Essex

February 5, 2018

About

Collaborative filtering

Content-based filtering

Implicit feedback

Conclusion

RECOMMENDER SYSTEMS

- ▶ We will discuss one of the most popular applications of data science
 - ▶ Recommender Systems
- ▶ Every website does it
- ▶ Recommender Systems match users with items
- ▶ Users under constant information overload
- ▶ Think songs, foods, drinks, movies, news

EXAMPLES

- ▶ This is even done offline in the service industry!
- ▶ Can you think of other examples?

COLLABORATIVE FILTERING

- ▶ Collaborative filtering is an effort to predict how products/items will be rated by a user, given previous ratings (from both the user and others)
- ▶ This prediction can help us recommend to the user only items that we think she will rate highly
- ▶ Latent Factor Models - Netflix Challenge (1M\$)- Simon Funk

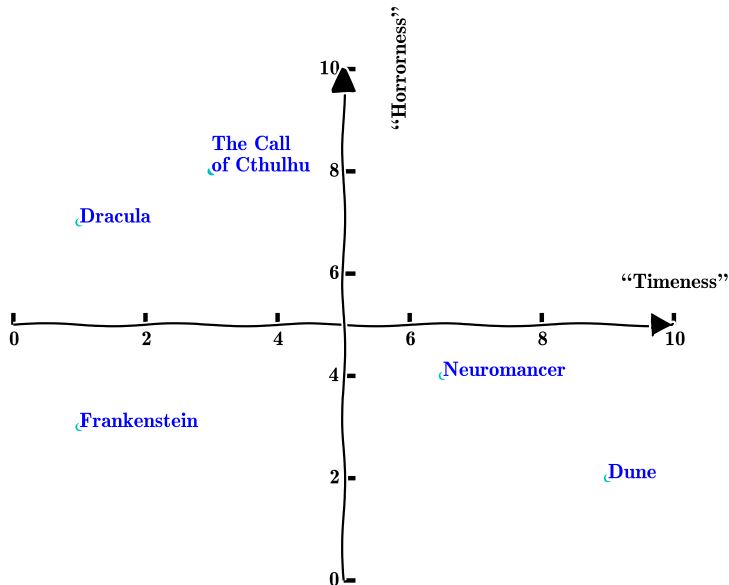
SAME SAMPLE DATA

	The Call of Cthulhu	Frankenstein	Dracula	Neuromancer	Dune
0	6	0	0	7	NaN
1	5	0	5	6	5
2	9	NaN	4	NaN	8
3	4	NaN	2	5	6
4	4	NaN	4	6	0
5	6	3	8	5	7
6	NaN	6	NaN	6	7
7	NaN	1	1	6	NaN
8	NaN	NaN	2	NaN	9
9	NaN	3	4	5	7

FACTORS

- ▶ We are going to base our predictions on “hidden” qualities of the items
- ▶ For example, food can have different levels of spiciness, a drink different levels of bitterness
- ▶ We term these qualities “factors”
- ▶ A sensible way of describing items would be to see them as a collection of “factors”
 - ▶ But our data is just ratings!
- ▶ Thus, our factors are “latent”, i.e. hidden!

EXAMPLE FACTORS FOR OUR DATA



FACTORS AND USER PREFERENCES

- ▶ Let's assume n factors
- ▶ We can encode factors as a real valued vector
 $\textit{item_factors}_i = [\phi_0, \phi_1, \dots, \phi_{n-1}]$
- ▶ For example “The Call of the Cthulhu” can be encoded as
 $\textit{item_factors}_0 = [3, 8]$
- ▶ Each user now can have preferences over factors, encoded as weights $\textit{user_preferences}_j = [w_0, w_1, \dots, w_{n-1}]$
- ▶ The weight vector contains user preferences, e.g.
 $\textit{user_preferences}_0 = [0.5, 0.8]$
- ▶ But we don't have any user weights nor any item factors - generate some random!

SOME RANDOM DATA

- Each row in *user_preferences* represents the preferences of a user, while each row in *item_factors* represents the factors of an item

```
array([[ 0.092,  0.783],  
       [ 0.78 ,  0.488],  
       [ 0.844,  0.062],  
       [ 0.68 ,  0.549],  
       [ 0.212,  0.43 ],  
       [ 0.961,  0.023],  
       [ 0.659,  0.31 ],  
       [ 0.92 ,  0.769],  
       [ 0.817,  0.452],  
       [ 0.834,  0.887]])  
  
array([[ 0.338,  0.519],  
       [ 0.69 ,  0.256],  
       [ 0.363,  0.93 ],  
       [ 0.004,  0.112],  
       [ 0.608,  0.104]])
```

$$rating[0][0] \leftarrow 0.437 = 0.092 * 0.338 + 0.783 * 0.519$$

Far away from the observed rating of 6

PREDICT RATINGS

- ▶ Example python code
- ▶ Still random values...

```
def predict_rating(user_row,item_row):  
    """ Predict a rating given a user_row and an item_row.  
    """  
    user_values = user_preferences[user_row]  
    item_values = item_factors[item_row]  
    return user_values.dot(item_values)
```

TRAINING FOR A SINGLE EXAMPLE

- ▶ *user_preferences* and *item_factors* have random values!
- ▶ Find the difference between the real and the predicted rating (“how far away am I from the goal?”)
- ▶ Multiply by small learning rate $\alpha = 0.0001$ (“Don’t take my measurement so seriously”)
- ▶ Move *user_preferences* and *item_factors* towards the correct value, following the negative of the gradient (“Let’s move towards the direction of the most abrupt change”)

TRAINING CODE

```
def train(user_row,item_row,rating,alpha = 0.0001):  
    """ Adapt the values of user_preferences and item_factors  
        to match the ones predicted by the users  
    """  
    err = alpha * (rating - predict_rating(user_row, item_row))  
    user_preferences[user_row] += err * item_factors[item_row]  
    item_factors[item_row] += err * user_preferences[user_row]
```

TRAINING USING ALL DATA

- ▶ “Latent Factors” because we have never really observed them, we can only infer them
- ▶ Loop over all *user_preferences* and *item_factors*
- ▶ Ignore cells with no value (“NaN” cells)
- ▶ Repeat until some criterion (in our case, 30,000 iterations)

TRAINING USING ALL DATA CODE

```
def sgd_svd(iterations = 30000):  
    """ Iterate over all users and all items and train for  
        a certain number of iterations  
    """  
    for i in range(0, iterations):  
        for user_row in range(0, user_preferences.shape[0]):  
            for item_row in range(0, item_factors.shape[0]):  
                rating = user_ratings[user_row][item_row]  
                if (not np.isnan(rating)):  
                    train(user_row, item_row, rating)
```

RECONSTRUCTING DATA / PREDICTING UNSEEN RATINGS

- Run `sgd_svd()` and print the updated tables

```
array([[ 1.705,  0.486],  
       [ 1.857,  0.484],  
       [ 2.373,  1.311],  
       [ 0.988,  1.495],  
       [ 2.519, -2.088],  
       [ 1.868,  1.235],  
       [ 1.367,  1.801],  
       [ 1.405,  0.628],  
       [ 0.133,  3.453],  
       [ 1.562,  1.235]])  
  
array([[ 2.713,  1.266],  
       [-1.125,  4.09 ],  
       [ 1.847,  0.514],  
       [ 3.09 ,  0.807],  
       [ 2.079,  2.522]])
```

$rating[0][0] \leftarrow 1.705 * 2.713 + 0.486 * 1.266 \approx 5.2$, not 6, but close!

VISUAL COMPARISON

- Calculate all predicted values and pretty print

	The Call of Cthulhu	Frankenstein	Dracula	Neuromancer	Dune
0	(6.000 5.209)	(0.000 0.029)	(0.000 3.397)	(7.000 5.699)	(nan 4.587)
1	(5.000 5.662)	(0.000 -0.046)	(5.000 3.699)	(6.000 6.208)	(5.000 4.956)
2	(9.000 8.112)	(nan 2.705)	(4.000 5.098)	(nan 8.443)	(8.000 8.198)
3	(4.000 4.564)	(nan 4.518)	(2.000 2.651)	(5.000 4.262)	(6.000 5.799)
4	(4.000 4.214)	(nan -9.302)	(4.000 3.425)	(6.000 6.127)	(0.000 0.017)
5	(6.000 6.617)	(3.000 3.004)	(8.000 4.100)	(5.000 6.756)	(7.000 7.003)
6	(nan 5.960)	(6.000 5.735)	(nan 3.473)	(6.000 5.593)	(7.000 7.508)
7	(nan 4.577)	(1.000 0.950)	(1.000 2.918)	(6.000 4.858)	(nan 4.397)
8	(nan 4.494)	(nan 12.716)	(2.000 2.010)	(nan 2.851)	(9.000 8.985)
9	(nan 5.769)	(3.000 3.336)	(4.000 3.523)	(5.000 5.774)	(7.000 6.389)

- For user 2, recommend “Neuromancer” and ignore “Frankenstein”
- Observe how reconstruction is not perfect - multiple reasons (e.g. data shuffling? mini-batches? more factors? more training)

OTHER OUTCOME SIGNALS

- ▶ We have used ratings
- ▶ But this is not the only possible outcome
- ▶ One can use other signals as well
 - ▶ User have searched for certain films
 - ▶ Users have clicked on certain films

HOW ABOUT ADDING KNOWN FACTORS/FEATURES?

- ▶ Until now we only had latent factors
- ▶ But latent factors arise only when you actually have a good number of $\langle user, item, ratings \rangle$ triplets for a user
 - ▶ What if the user just joined the website?
- ▶ What if you don't have any?
 - ▶ Or what if you have further observations about a user

ITEM DESCRIPTIONS

- ▶ Instead of based only on latent factors, we can base our predictions on known observations
- ▶ For example each book can have:
 - ▶ Genre
 - ▶ Data published
 - ▶ Age of intended audience
 - ▶ Author
- ▶ Each film can have
 - ▶ Genre
 - ▶ Director
 - ▶ Age of intended audience

USER DESCRIPTIONS

- ▶ But we might have data collected about a user as well
 - ▶ Age
 - ▶ Sex
 - ▶ Country of birth
 - ▶ Native language
- ▶ All kinds of data

EXPLICIT KNOWLEDGE

- ▶ You ask the user questions explicitly
 - ▶ What kind of books do you like?
 - ▶ What is the maximum price you would pay for a book?

CONTENT BASED FILTERING

- ▶ One can use all features defined on the user and the item
- ▶ Create a classifier and do the predictions using the classifier
- ▶ We have very few samples
 - ▶ So we are going to use a linear classifier

ITEM FEATURE MATRIX

Feature 0: First critic score of the book

Feature 1: Second critic score of the book

	Critic0	Critic1
0	0.3	0.9
1	0.9	0.3
2	0.6	0.4
3	0.2	0.1
4	0.7	0.8
5	0.9	0.1

USER FEATURE MATRIX

Feature 0: Male/Female

Feature 1: Over 60

	Sex	Over60
0	1.0	0.0
1	0.0	1.0
2	0.0	0.0
3	1.0	0.0
4	0.0	1.0
5	0.0	0.0
6	0.0	0.0
7	1.0	0.0
8	0.0	1.0
9	1.0	0.0

COMBINING THE FEATURES

- We need to build a set of features for training for each person/item combo

	Sex	Over60	key	user_id	Critic0	Critic1	item_id	rating
0	1.0	0.0	0	0	0.3	0.9	0	8.0
1	1.0	0.0	0	0	0.9	0.3	1	2.0
3	1.0	0.0	0	0	0.2	0.1	3	5.0
4	1.0	0.0	0	0	0.7	0.8	4	4.0
0	0.0	1.0	0	1	0.3	0.9	0	3.0
1	0.0	1.0	0	1	0.9	0.3	1	2.0
3	0.0	1.0	0	1	0.2	0.1	3	7.0
4	0.0	1.0	0	1	0.7	0.8	4	7.0
0	0.0	0.0	0	2	0.3	0.9	0	9.0
2	0.0	0.0	0	2	0.6	0.4	2	7.0
3	0.0	0.0	0	2	0.2	0.1	3	8.0
4	0.0	0.0	0	2	0.7	0.8	4	5.0
2	1.0	0.0	0	3	0.6	0.4	2	7.0
3	1.0	0.0	0	3	0.2	0.1	3	8.0
4	1.0	0.0	0	3	0.7	0.8	4	9.0
1	0.0	1.0	0	4	0.9	0.3	1	1.0
2	0.0	1.0	0	4	0.6	0.4	2	8.0
3	0.0	1.0	0	4	0.2	0.1	3	3.0

TEST SET

- We are looking to predict this:

	Sex	Over60	key	user_id	Critic0	Critic1	item_id	rating
2	1.0	0.0	0	0	0.6	0.4	2	NaN
2	0.0	1.0	0	1	0.6	0.4	2	NaN
1	0.0	0.0	0	2	0.9	0.3	1	NaN
0	1.0	0.0	0	3	0.3	0.9	0	NaN
1	1.0	0.0	0	3	0.9	0.3	1	NaN
0	0.0	1.0	0	4	0.3	0.9	0	NaN
3	0.0	0.0	0	5	0.2	0.1	3	NaN
4	0.0	0.0	0	5	0.7	0.8	4	NaN
2	0.0	0.0	0	6	0.6	0.4	2	NaN
2	0.0	1.0	0	8	0.6	0.4	2	NaN
1	1.0	0.0	0	9	0.9	0.3	1	NaN

CODE - PANDAS MAGIC!!!

```
user_ratings_df = pd.read_csv("user_ratings.csv")
user_features_df = pd.read_csv("user_features.csv")
item_features_df = pd.read_csv("item_features.csv")

user_features_df["key"] = 0
user_features_df["user_id"] = range(0,user_features_df.shape[0])
item_features_df["key"] = 0
item_features_df["item_id"] = range(0,item_features_df.shape[0])

merged_df = pd.merge(user_features_df, item_features_df,left_index=True,on="key")

merged_df["rating"] = map(lambda ids: user_ratings_df.values[ids[1]][ids[2]],
                          merged[["user_id", "item_id"]].itertuples())

train = merged_df.dropna()

test = merged_df[merged.isnull().any(axis=1)]
```

HYBRID SYSTEMS

- ▶ Can we merge the two approaches?
 - ▶ Of course we can - various ways of merging
- ▶ We will just augment collaborative filtering with standard features for now
- ▶ We can add the features as parts of the variables we are going to learn

REGULARISATION

- ▶ We will now add a penalty to features depending on what they are
- ▶ Penalty strong for latent features
- ▶ But could be the other way around
- ▶ Weight decay / l_2 regulariser
 - ▶ $w_i = w_i - \alpha(y_p - y)(\phi_i + \lambda w_i)$

CODE - PREDICT

```
def predict_rating(user_id,item_id):  
    """ Predict a rating given a user_id and an item_id.  
    """  
  
    user_preference = latent_user_preferences[user_id]  
    item_preference = latent_item_features[item_id]  
  
    user_score = user_features_weights[user_id].dot(user_features[user_id])  
    item_score = item_features_weights[item_id].dot(item_features[item_id])  
  
    return user_preference.dot(item_preference) + user_score + item_score
```

CODE - TRAIN

```
def train(user_id, item_id, rating,alpha = 0.0001,
          latent_feature_weight_decay = 0.1,
          user_weight_decay = 0.01,
          item_weight_decay = 0.001):

    prediction_rating = predict_rating(user_id, item_id)
    err = ( prediction_rating - rating );

    user_pref_values = latent_user_preferences[user_id][:]
    latent_user_preferences[user_id] -= alpha *
        err * ( latent_item_features[item_id] +
                latent_feature_weight_decay*
                latent_user_preferences[user_id])

    latent_item_features[item_id] -= alpha *
        err * ( user_pref_values +
                latent_feature_weight_decay*latent_item_features[item_id])

    user_features_weights[user_id] -=alpha * err * ( user_features[user_id] +
        user_weight_decay* user_features_weights[user_id])
    item_features_weights[item_id] -=alpha * err * ( item_features_weights[item_id] +
        item_weight_decay* item_features_weights[item_id])

    return err
```


REMOVING LATENT FACTORS

- ▶ What will happen to the example above if we remove all latent factors?
 - ▶ ... and base our decisions only in known features
- ▶ Factors vs features

EVALUATING

- ▶ Cross-validation again!
- ▶ We have used all the data in our examples
 - ▶ Is this correct?
 - ▶ We have also evaluated the system using MSE - this is not entirely correct either
- ▶ What would be the proper way of evaluating the system?

IMPLICIT FEEDBACK

- ▶ When you are recommending books or food user preferences user likes/dislikes something
- ▶ How about if you are recommending
 - ▶ News?
 - ▶ Adverts?
- ▶ You are limited to clicks!
- ▶ But they are not proper feedback
 - ▶ User might not have seen something

NO NEGATIVE FEEDBACK REGIMES

- ▶ Not clicking is not negative feedback
- ▶ Create a preference matrix - 1 if user has clicked, 0 if she hasn't
- ▶ Weight the importance of each example by $c(i, j) = 1 + \alpha r(i, j)$
 - ▶ $r(i, u)$ is the number of clicks

CONTEXTUAL BANDITS

- ▶ You can personalise the above scenario even further
- ▶ Send the user some random examples (e.g. news)
 - ▶ With let's say 0.2 probability
- ▶ Next week more on this...

METRICS

- ▶ Serendipity
 - ▶ Our predictions might not be surprising - you don't just need present the user with new items, but also with items that are sufficiently different than rated
- ▶ Coverage
 - ▶ Some items are not bought often - they should be recommended as well
- ▶ We have measured MSE, but ultimately this is a ranking problem
 - ▶ We can use ranking metrics, e.g. Mean Average Precision
- ▶ Diversity
 - ▶ Some items are way too similar to each other - should they be presented to the user?

CONCLUSION

- ▶ We have seen various instances of recommender systems
- ▶ Again, there are far more complex models
- ▶ But the examples here should have given you a good view about what is out there
- ▶ Also, pandas
 - ▶ Bring your data to a format that is usable by relevant libraries!