Recommender Systems: Content-based Filtering

AAA-Python Edition

Plan

- 1- Introduction
- 2- Content Based Filtering with Decision Trees
- 3- User profiles Learning with Decision Trees
- 4- Make predictions with the decision tree
- 5- Nearest Neighbor Method
- 6- Polynomial Regression

Concept

1- Introduction

- As said in [Francesco et al., 2011]**, Content Based** recommendation systems try to recommend items **similar** to those a given user has liked in the past.
- The recommendation for an **item** to a **user** is based on the **actual features** that **this item** has. And, on actual feedback and reviews that **this user** has already done.
- The system is build around **3** components:
  - **The item description**: each item is described by a set of attributes (features)
  - **The user profile**: each user is described by a model generated from the features of the items rated by that user
  - **A matching strategy**: how to match up the user's profile attributes with an item attributes

[By Amina Delali]

Process

Analyze content

Learn users profiles

Build a filtering component

We preprocess the data to construct a **structured items description**

We use the users **preferences** for **items** to construct a user profile

We use the user profile to **recommend new items** by **matching** the representation of the user's profile with the representation of the new items. **Feedback** of the user on these **recommendations** are used to **enhance** the user profile.

[By Amina Delali]

1- Introduction

Learning User Profiles

- Different approaches exist to construct a user profile. They are based on techniques of **text classification**:
  - **Probabilistic methods and Naive Bayes**: a probabilistic text classification approach based on the Naive Bayesian Classifier (see Week4 lesson 3 for more details)
  - **Rocchio's algorithm:** is a relevant feedback algorithm. It refines the recommendations by using the feedback of users on these recommendation. The update process takes into account relevant and non relevant recommended items.
  - **Decision Trees Classifier** (see Week5 lesson 1)
  - **Nearest neighbors algorithm** (this week, lesson 1)
  - **Linear Classifiers (**week 4 lessons)

2- Content Based Filtering with Decision Trees

Concept

- Each user profile will be represented by a **Decision Tree Classifier**. The tree will learn from the description of **items** he already reviewed

- The **labels** will be the **ratings** (or **like** and **dislike**) the **user** had given to these items.

- The resulting Tree will later used to:

  - predict the review (rating) of a new item.

  - make a list of recommendations to that user based on the predictions on a list of items.

Items Description

2- Content Based Filtering with Decision Trees

- In general, the items are described by **text**. So, in order to use them, we have to vectorize our text data. (See week 4 lesson1)
- We will built a movie recommender for **one** user, using the data available at: The Movies Dataset | Kaggle

```
import pandas as pd
from pandas import DataFrame as DF

myDF = pd.read_csv("AAA-Ped-Week7/A3P-w7-movies_metadata.csv")
print(myDF.shape)
myDF.head(1)
```

```
(45466, 24)
```

The file contain a description for **45466** movies in **24** columns

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	862	tt0114709

[By Amina Delali]

The Labels

- We will use the **ratings** that users gave to these movies as labels.
- We will select only the ratings made by the user with the **id_user = 2**

```
myDFR = pd.read_csv("AAA-Ped-Week7/A3P-w7-ratings_small.csv")
print(myDFR.shape)
myDFR.head(1)
```

The data contains 1000004 ratings

```
(100004, 4)
```

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144

```
myDFU1= myDFR[myDFR["userId"] == 1]
print("The user 1 rated: ",myDFU1.shape[0]," movies")
myDFU1.head(2)
```

```
The user 1 rated:  20  movies
```

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179

[By Amina Delali]

Preparing the Data

- In this phase we will merge the ratings done by the user **2** with the description of the movies he rated. (More details about **merging** operations are available in week 3 lesson 1)

```python
# First we have to convert the movieId in the user
# dataframe into string
def myMap(x):
  return str(x)
myDFU1.loc[:,"strId"]=myDFU1.loc[:,"movieId"].apply(myMap)
myDFU1.head(3)
```

	userId	movieId	rating	timestamp	strId
20	2	10	4.0	835355493	10
21	2	17	5.0	835355681	17
22	2	39	5.0	835355604	39

The id type in the user dataframe was int, and in the movies dataframe , it was a string. So , we had to convert one of them before making the merge

```python
# merge the 2 dataframes, keep only the rows corresponding to movies id
# that are in both frames
mergedDF = pd.merge(myDFU1,myDF,left_on="strId",right_on="id",how="inner")

print(mergedDF.shape)
mergedDF.head(1)
```

```
(58, 29)
```

	userId	movieId	rating	timestamp	strId	adult	belongs_to_collection

[By Amina Delali]

3- User profiles Learning with Decision Trees

Extracting Features

```
# now we will selcet only the columns of the attributes
# that we are intrested in
myDataF = mergedDF[["overview","rating"]]
myDataF.head(1)
```

	overview	rating
0	Adèle and her daughter Sarah are traveling on ...	5.0

We will user only the overview of the movie as description attribute. The rating is for the classification labels

```
# Features extractions
from sklearn.feature_extraction.text import TfidfVectorizer
myVectorizer = TfidfVectorizer()
myX = myVectorizer.fit_transform(myDataF["overview"])
print("The size of the features matrix is ",myX.shape)
#the lables for learning
myY= myDataF.rating.values

DFX =DF(myX.toarray(),columns= myVectorizer.get_feature_names())
DFX.head(0)
```

We apply a TF-IDF transformation on the overview attribute

```
(58, 1267)
```

1973	1980	1985	2000	...	worn	writer	yearnings	years	yet	york	you	young

Words available in the overviews

[By Amina Delali]

3- User profiles Learning with Decision Trees

Learning the User’s Profile

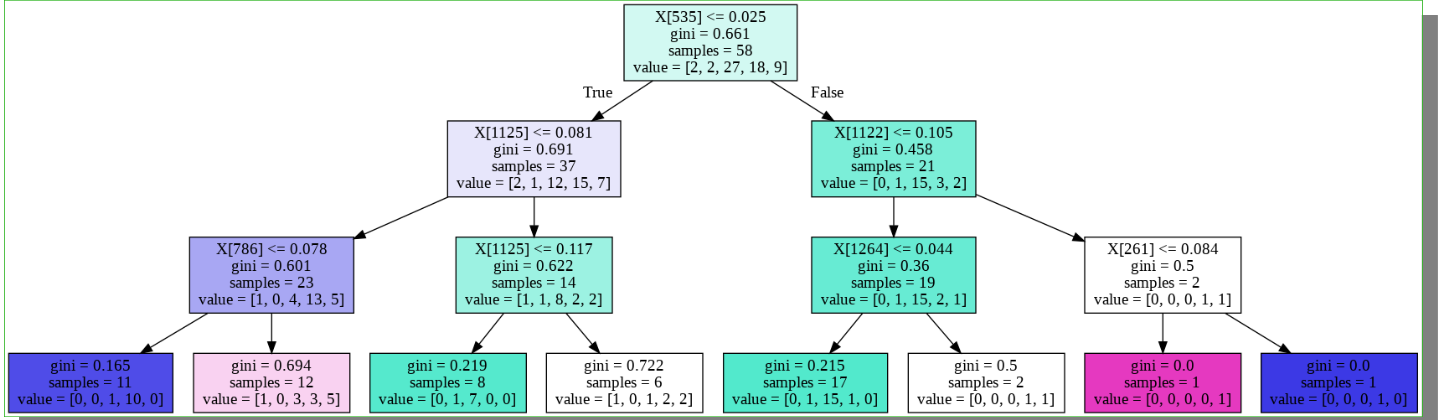
```python
from sklearn.tree import DecisionTreeClassifier as DTC
from sklearn.model_selection import cross_validate as c_v

myDTC = DTC(max_depth=3)
scores= c_v(myDTC,myX,myY,cv=3, scoring=["neg_mean_absolute_error"])
print("the mean of MAE of the test folds: ",np.abs(scores["test_neg_mean_absolute_error"]).mean())
```

```
the mean of MAE of the test folds:  0.8759259259259259
```

```python
#fit the model
myDTC.fit(myX,myY)
myDTC.classes_
```

```
array([1., 2., 3., 4., 5.])
```

The tree representing the user’s model



[By Amina Delali]

4- Make prediction with the decision tree

Not rated movies

- We have to create the list of the movies that the user **2** didn't rate. Then we will select only **30** movies.
- We have also to select only the features represented in the user's profile.

```
#All movies
allMov = myDF.id.values
# Movies rated by 2
By2  = myDFU1.strId.values
# Movies not rated by 2
notBy2 = [i for i in allMov if i not in By2]
DFnotB2 = DF(notBy2,columns=["idNB2"])
# description of movies not rated by 2
myDFnotB2 = pd.merge(DFnotB2,myDF,left_on="idNB2",right_on="id",how="inner")
```

We selected only **30** movies for memory issues

```
# select the attribute
myFinalDF = myDFnotB2["overview"]
# drop nan values
myFinalDF.dropna(inplace=True)
#extract the features, for only 30 movies
myX2 = myVectorizer.fit_transform(myFinalDF.values)
DFX2 = DF(myX2[:30,:].toarray(), columns= myVectorizer.get_feature_names())
```

[By Amina Delali]

4- Make prediction with the decision tree

Not rated movies (suite)

```
from pandas import DataFrame as DF
myNewDFX = DF()

# The dataframe must contain only the user's profile columns
for i in DFX.columns:
  if i in DFX2.columns:
    myNewDFX[i]=DFX2[i]
  else:
    myNewDFX[i]=0
```

1973	1980	1985	2000	...	worn	writer	yearnings	years	yet	york	you	young
0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

The items description used for the prediction must:

- Contain the same features as those used by tree classifier to model the user's profile
- The features must be ordered the same way.

[By Amina Delali]

4- Make prediction with the decision tree

Make predictions

- We will make predictions for the first **30** movies in **myX2**.
- Then, we will sort these predictions, and select the **10** first ratings

```
The 10 biggest ratings:
1 -  Cutthroat Island  ( 5.0 )
2 -  Ace Ventura: When Nature Calls  ( 5.0
3 -  Copycat  ( 4.0 )
4 -  Tom and Huck  ( 4.0 )
5 -  Casino  ( 4.0 )
6 -  Sense and Sensibility  ( 4.0 )
7 -  Four Rooms  ( 4.0 )
8 -  Heat  ( 4.0 )
9 -  Powder  ( 4.0 )
10 -  Grumpier Old Men  ( 4.0 )
```

```python
# predict the class of the ratings
myPredictions= myDTC.predict(myNewDFX)
# sort the predictions by descending order
indSort = np.argsort(myPredictions)[::-1]
# print the 10 biggest scores
print("The 10 biggest ratings:")
for i in range(10):
  j= indSort[i]

  print(i+1,"- ",myDFnotB2.iloc[j,21]," (",myPredictions[j],")")
```

Since we didn’t change the order of the rows of the dataframe corresponding to the “not rated” movies ‘myDFnotB2) , and since we selected only the first elements sequentially, the order the predictions array and the myDFnotB2 frame is the same.

[By Amina Delali]

5- Nearest Neighbor Method

Concept

- After preparing the descriptions of the items rated by one user. And in order to predict a review on a new item (by that user):

  - Compute the **similarity** between the new item and all the rated item
  -
  - Select the **nearest** or the **k nearest neighbors** items using the computed similarities

  - aggregate the ratings of the selected items.

- In the case of vectorized text attributes, the **cosine similarity measure** could be used.

[By Amina Delali]

5- Nearest Neighbor Method

Compute similarities

- We will select one item from the not rated items (of the previous section), then we will compute its **cosine similarities** with all the movies rated by the user **2**

```
# the movie we selected
myDFnotB2.iloc[13,21]
```

```
'Nixon'
```

```
# compute the similarities
from sklearn.metrics.pairwise import cosine_similarity

similarities = cosine_similarity(myNewDFX.iloc[13].reshape(1,-1),DFX )
similarities
```

```
array([[0.03938174, 0.07689689, 0.06380002, 0.07007084, 0.01557274,
        0.08658868, 0.07756476, 0.09133323, 0.13468837, 0.11750274,
        0.07958574, 0.07407981, 0.11547842, 0.09670878, 0.18569619,
        0.06862389, 0.01040764, 0.12456263, 0.0549071 , 0.0417849 ,
        0.16781957, 0.11108256, 0.11152991, 0.0412832 , 0.14656956,
        0.04338626, 0.05396072, 0.08042772, 0.11006106, 0.05736534,
        0.0993555 , 0.08228833, 0.11171322, 0.0732958 , 0.07650725,
```


[By Amina Delali]

Make predictions

5- Nearest Neighbor Method

```
# we will sort the similarities in a descending order
# the select the 10 first neighbors
simOrd = np.argsort(similarities[0])[::-1]

neighbors = myY[simOrd[:10]]
# aggregate the review
print("The predicted rating is: ", np.round(neighbors.mean(),2))
```

```
The predicted rating is:  2.9
```

```
# the indices in simOrd correspond to the order
# in similarities and in mergedDF
print("The movies (rated by user2) the most similar to the movie 'Nixon':")

for i in range(10):
  j = simOrd[i]
  print(i+1,"-",mergedDF.iloc[j,25]," (",np.round(similarities[0][j],2),")")
```

```
The movies (rated by user2) the most similar to the movie 'Nixon':
1 - Wag the Dog  ( 0.19 )
2 - Stand by Me  ( 0.19 )
3 - Big Fish  ( 0.17 )
4 - Batman Begins  ( 0.17 )
5 - The Science of Sleep  ( 0.15 )
6 - Star Trek IV: The Voyage Home  ( 0.13 )
7 - The Last Samurai  ( 0.13 )
8 - Cat on a Hot Tin Roof  ( 0.12 )
9 - A Clockwork Orange  ( 0.12 )
10 - Rebecca  ( 0.12 )
```

[By Amina Delali]

References

- [Francesco et al., 2011] Francesco, R., Lior, R., Bracha, S., and Paul B., K., editors (2011). Recommender Systems Handbook. Springer Science+Business Media.

- [Kaggle,] Kaggle. The movies dataset. https://www.kaggle.com/rounakbanik/the-movies-dataset.

- [Pazzani and Billsus, 2007] Pazzani, M. J. and Billsus, D. (2007). Content-based recommendation systems. In The adaptive web, pages 325–341. Springer

Thank you!

FOR ALL YOUR TIME