ASHISH RANA Follow

Apr 18, 2019 · 11 min read · ✦ · Listen

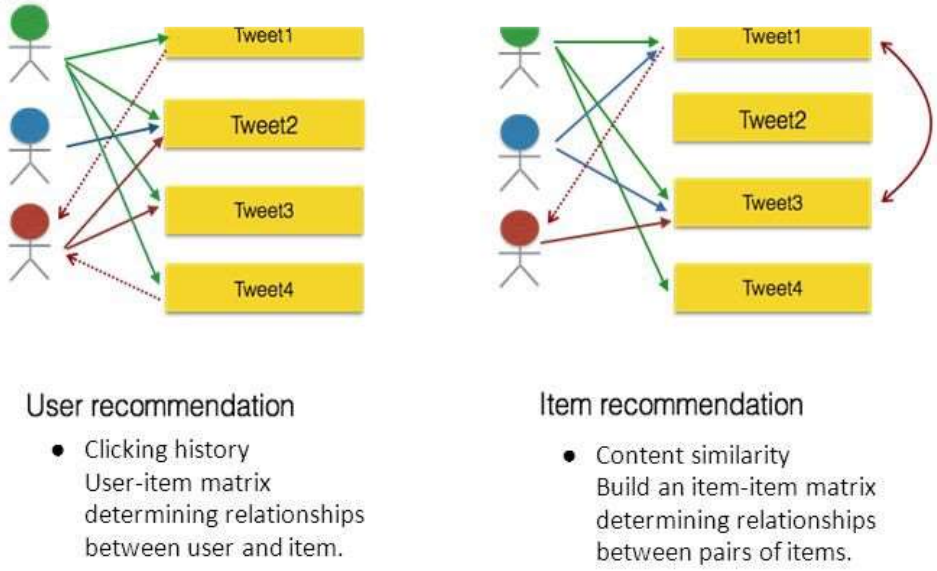


# Build Your Own Clustering Based Recommendation Engine in 15 minutes !!

Recommendation engines are one of the most popular application of machine learning techniques in current internet age. These are extensively used in e-commerce websites for recommending similar products and on movie recommender sites. They are responsible for generating various custom tailored news suggestions for us. Which will drive more content engagement from user leading better user experience and more revenue for organization. Hence, they are of extreme importance in today's industry.

Recommendation engines basically filters the data and recommend most relevant results to users. These results are recommended in such manner that likelihood of interest in results in maximum. Now, all the recommendation engines have user data and their history available with them for creating their filtering algorithms to work. Which eventually helps them generate very accurate recommendations for each unique user.

base




User-based filtering is based on history of users and similarity b/w them from their purchase histories for example. But, Item-based recommendations are based on content based similarity. Like, “how many times few items are bought together”. Next time, most frequent of these purchases will be recommended together.

In case of collaborative filtering “User Behavior” is cashed in for recommending items. These recommendations can be generated with user-user similarity or on the basis of item-item similarity. And on the basis of this similarity measure the suggestions are provided to user. But, let’s consider a scenario where no user data is available to us and we still have to recommend items to user.

## What to do, without the user data ? How will our recommendation engine work now ?

The problem of generating recommendations now get transformed simply into a clustering like problem. Where the similarity measure is based on “How close two items are, while generating recommendations ?”. The measure for generating recommendation will be on the basis of similarity of two items like vector distance between these items. We will be carrying our discussion on this for online course text data from Pluralsight. Let’s make a recommendation engine based only on item data

and see furt ution. We will discuss entioned sequence. In order to save time you can directly refer to project repository and follow along the elaborative README.md file. Also, can run direct utility scripts for each and every module mentioned.

1. *Introduction: Know your Data*
2. *Architectural Design: Build a Utility Tool*
3. *Pre-processing Steps*
4. *Problem Discussion, Model Training and Optimizations*
5. *Working Recommendation System*
6. *Conclusion & Future Improvements with Topic Modelling (LDA specifically)*

> Super Time Saver Tip for Pros: Just open github repository of project and follow along README.md file and run the code 😉

## Introduction: Know your Data

The data that is used for the project is list and description of courses hosted on Pluralsight website. For, obtaining course data simply run below mentioned ReST API query. But, for obtaining user enrollment data along with it let's say for a collaborative filter based engine.

First, obtain the ReST api-token as mentioned in documentation and after that make the ReST query to fetch data about all the courses on this website and respective users enrolled in it. This key is required if you want to obtain user related data. Otherwise for obtaining simple course related data we can just write the following ReST query as

```
# Output                                                    elow
mentione(

CourseId,CourseTitle,DurationInSeconds,ReleaseDate,Description,Assessm
entStatus,IsCourseRetired

abts-advanced-topics,BizTalk 2006 Business Process
Management,22198,2008-10-25,"This course covers Business Process
Management features in BizTalk Server 2006, including web services,
BAM, hosting, and BTS 2009 features",Live,no
abts-fundamentals,BizTalk 2006
...
```

In this article we limit ourselves to course data only for engine construction. Otherwise, the approach will very similar to other recommendation engine articles out there . By taking a look at this data we observe following points that are quite important while training a model. You, can open *Courses.csv* file and make the following observation by yourself also.

1. The textual description of course data is present for CourseId, Course Title and Course Description column. Hence, these columns are of interest while constructing our recommendation engine. With textual data from these columns we will be able to construct word vectors that'll be used by our model while predicting results. Also, most of the information is present in *'Description'* column only. Hence, courses with no description will be dropped off from training.

2. 'IsCourseRetired' column gives description about current status of course on website i,e. is the course currently available on website or not. Hence, we don't want to recommend retired courses from our trained model. But, we can definitely use them in our training data.

3. Also, carrying on discussion about pre-processing of this data. There are clearly some extra '-' tokens, different cases and stopwords present in data. We'll pre-process our text accordingly and focus on Noun/Noun-Phrases only.

In next section we'll discuss the basic architecture of this recommendation utility

## Architectu[illegible]

The diagram below clearly states our pipeline for this data science project. Please, have a look at it before reading further in left to right manner.

**Three main components:** 1. Pre-process & Train; 2. Optimizations; 3. Recommendation Utility Tool

This utility tool is mainly divided into three components and we'll discuss these components in detail in further sections to come. Mainly, we'll train the model and optimize it to reduce the error. After that, we'll code the utility tool which will generate the recommendations based on input queries of unique course ids.

With above architecture of tool in mind, let's move to pre-processing step and start working on data ingestion step for our model.

## Pre-processing Steps

we'll, you'll ... also eliminate st... id, title in appropriate manner. Refer, the code snippet below to follow along above mentioned steps.

```
import pandas as pd

# 1. read data, from source
# "Courses.csv" file has been renamed
course_df = pd.read_csv("data/courses.csv")

# 2. drop rows with NaN values for any column, specifically
'Description'
# Course with no description won't be of much use
course_df = course_df.dropna(how='any')

# 3. Pre-processing step: remove words like we'll, you'll, they'll
etc.
course_df['Description'] = course_df['Description'].replace({"'ll": "
"}, regex=True)

# 4. Another Pre-preprocessing step: Removal of '-' from the CourseId
field
course_df['CourseId'] = course_df['CourseId'].replace({"-": " "},
regex=True)

# 5. Combine three columns namely: CourseId, CourseTitle, Description
comb_frame = course_df.CourseId.str.cat("
"+course_df.CourseTitle.str.cat(" "+course_df.Description))

# 6. Remove all characters except numbers & alphabets
# Numbers are retained as they are related to specific course series
also
comb_frame = comb_frame.replace({"[^A-Za-z0-9 ]+": ""}, regex=True)
```

After carrying out basic cleaning steps for the above data, *'comb_frame'* contains all the necessary word descriptions related to a course. After, that let's move onto to vectorization of this text and train our model.

## Problem Discussion, Model Training and Optimizations

For this, we ocument. It is a statistica related to number of times word appear in a corpus but is offseted by frequency of words in corpus.

***Tf*** in ***tf-idf*** weight measures frequency of terms in a document. And ***idf*** measure importance of that given term in given corpus. This can be inferred from formulas mentioned below.

```
TF(t) = (Number of times term 't' appears in a document) / (Total
number of terms in the document)

IDF(t) = log_e(Total number of documents / Number of documents with
term 't' in it)
```

We will use scikit learn to convert our textual data into a vector matrix product as specified in above formula. Follow along with below code snippet for this conversion.

```
# Create word vectors from combined frames
# Make sure to make necessary imports

from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(comb_frame)
```

After this, we can feed this data straight into our k-means learning algorithm. But, we'll be needing idea value of ***'k'*** for our k-means algorithm about which we have had no discussion as such. We can use value k=8 for starters, as Pluralsight is having categories for eight different types of courses and check prediction abilities of our model trained accordingly. Follow along, the below mentioned code snippet for this.

```
true_k =

# Runnin                                                    aximum
iterations are 500
model = KMeans(n_clusters=true_k, init='k-means++', max_iter=500,
n_init=15)
model.fit(X)
```

We can observe top words from each cluster to see whether clusters formed are good qualitatively or they need improvement in some sense. Run the below mentioned snippet for observing top words in each cluster formed.

```
# Top terms in each clusters.

print("Top terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()
for i in range(true_k):
    print("Cluster %d:" % i),
    for ind in order_centroids[i, :15]:
        print(' %s' % terms[ind]),
    print
```

After observation of these words you might noticed that all the clusters formed are not appropriate and some course categories are being repeated over multiple clusters( Refer, README.md file for this). That is still fine (*for now* 😉), our model has sub-divided wider course categories with huge number of courses into other sub-categories also. Hence, cardinality issue for number of courses for given category is exposed and our model was unable to cope from it.

We can see that, sub-division categories graphic art, movie design, animation formed from parent 'creative-professional' category. This sub-categories are formed as data among course categories was not equi-distributed i.e. issue with cardinality of data. Hence, course categories like 'business-professional' with small number of courses got lost in our ideal assumption for **k** being equal to 8. It can easily happen as business

courses. For on problem with error m ich, we'll use ***'elbow-test'*** method for finding ideal value of ***k***. *The idea is whenever a sharp drop in error comes for a given value of* ***'k'****, that value is good enough for forming clusters.* These formed clusters will have sharp minima in error and will give satisfactory solution for our model. Follow along with below mentioned code for carrying out elbow-test on our data.

```
# Continuing after vectorization step

# data-structure to store Sum-Of-Square-Errors
sse = {}

# Looping over multiple values of k from 1 to 30
for k in range(1, 40):
    kmeans = KMeans(n_clusters=k, init='k-means++',
max_iter=100).fit(X)
    comb_frame["clusters"] = kmeans.labels_
    sse[k] = kmeans.inertia_

# Plotting the curve with 'k'-value vs SSE
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.ylabel("SSE")
# Save the Plot in current directory
plt.savefig('elbow_method.png')
```

After running the above code we got the following graph, on the basis of which we trained our model for k=30. And achieved relatively better clusters for our recommendation engine tool.

Slope is drastically diminishing after the value of k=30. Hence, we'll opt for this value for our model.

In the end, let's save our model and move on to our recommendation utility script design and discuss future improvement approaches also. All these mentioned snippets are available in the form of model_train.py script you can refer it for direct execution. But, before that do extract the courses.csv data file and go through README.md.

```
# Save machine learning model
filename = 'finalized_model.sav'
pickle.dump(model, open(filename, 'wb'))
```

## Working Recommendation System

We will create few utility functions for this recommendation module. A cluster_predict function which will predict the cluster of any description being inputted into it. Preferred input is the 'Description' like input that we have designed in comb_frame in

```
pred
retu
```

After, that we'll assign categories to each course based on their description vector in a new dataframe column namely *'ClusterPrediction'*. See below.

```
# Create new column for storing predicted categories from our trained
model.
course_df['ClusterPrediction'] = ""
```

We will store this cluster category score for a data-frame having only live courses i.e. the courses with 'no' live entry are dropped. After, that we'll run our predict function utility for each course in data-frame and store cluster categories. These, stored categories will be matched in future with the input query and its predicted category for generating recommendations.

```
# load the complete data in a dataframe
course_df = pd.read_csv("data/courses.csv")

# drop retired course from analysis. But, courses with no descriptions
are kept.
course_df = course_df[course_df.IsCourseRetired == 'no']

# create new column in dataframe which is combination of (CourseId,
CourseTitle, Description) in existing data-frame
course_df['InputString'] = course_df.CourseId.str.cat("
"+course_df.CourseTitle.str.cat(" "+course_df.Description))

# Create new column for storing predicted categories from our trained
model.
course_df['ClusterPrediction'] = ""

# Cluster category for each live course
course_df['ClusterPrediction']=course_df.apply(lambda x:
cluster_predict(course_df['InputString']), axis=0)
```

transformed course.

```
def recommend_util(str_input):

    # match on the basis course-id and form whole 'Description' entry
out of it.
    temp_df = course_df.loc[course_df['CourseId'] == str_input]
    temp_df['InputString'] = temp_df.CourseId.str.cat("
"+temp_df.CourseTitle.str.cat(" "+temp_df['Description']))
    str_input = list(temp_df['InputString'])


    # Predict category of input string category
    prediction_inp = cluster_predict(str_input)
    prediction_inp = int(prediction_inp)

    # Based on the above prediction 10 random courses are recommended
from the whole data-frame
    # Recommendation Logic is kept super-simple for current
implementation.

    temp_df = course_df.loc[course_df['ClusterPrediction'] ==
prediction_inp]
    temp_df = temp_df.sample(10)

    return list(temp_df['CourseId'])
```

Test your trained recommendation engine with the given following queries. You, can also add your queries also from picking up course-ids from courses.csv.

```
queries = ['play-by-play-machine-learning-exposed', 'microsoft-
cognitive-services-machine-learning', 'python-scikit-learn-building-
machine-learning-models', 'pandas-data-wrangling-machine-learning-
engineers', 'xgboost-python-scikit-learn-machine-learning']

for query in queries:
    res = recommend_util(query)
    print(res)
```


but gives the [illegible] hms. Also, recommend [illegible] ch like a top scored based recommendation approach can be adopted as improvements. Currently, course-id is acting as sole input instead a better natural language inputs should be there. But, these are implementation based improvements only.

Fundamentally, for future improvements the category assign mechanism and models used for training can be changed. More, advanced and sophisticated mechanisms from topic modelling like Latent Dirichlet Allocation(LDA) can be adopted. Topic modelling is statistical branch of NLP which extracts out abstracts from collection of documents. We'll use LDA, which will assign a particular document to a particular topic and a real number weight score that will be associated with words for corresponding topic.

> Just, run lda_train.py for understanding LDA implementation in detail and the comments/console output will explain everything about steps being executed.

These assigned topics and their associated score with words can act as prediction logic basis for above discussed *cluster_prediction* function. But, these predictions will more precise than any of the recommendations by k-means clustering algorithm currently being generated. A gensim based implementation for LDA is available here on same github repository. It's recommendation utility script is currently not added, you can try that as homework.

Hope, you enjoyed reading it and got a small hands on project for data-science. In case of any improvements do make a PR or open an issue on github.