A Recommendation System Using BigML's Topic Model Algorithm

Description

In this notebook, we will build a recommendation system using BigML's Topic Modeling, more specifically, we will take advantage of the topic distributions the model includes. This workflow will build a recommendation system for any data that has a title and a description. For this specific notebook, we will use a dataset that contains movie titles and descriptions.

Import Modules

- You will first need to install a number of modules in order to follow along with this notebook.
- Most of these packages, such as numpy and pandas, are available using Anaconda (https://conda.io/docs/user-guide/install/index.html).
- For the machine learning pipeline, we will be making use of the BigML Python bindings (https://bigml.readthedocs.io/en/latest/).
- To optimize our nearest neighbour search for our large high-dimensional topic data set, we will be using locality-sensitive hashes built from Nearpy (https://github.com/pixelogik/NearPyNearpy). To install Nearpy, one can use the line "pip install NearPy".
- To construct our distance formula that will be used in our locality-sensitive hashes, we will use entropy from scipy and Distance from Nearpy.
- To retrieve our locally stored Topic Model, we will be using pystemmer.

In [1]:
```
import sys
!{sys.executable} -m pip install NearPy
```

```
Requirement already satisfied: NearPy in /anaconda3/lib/python3.6/site-pa
ckages (1.0.0)
Requirement already satisfied: bitarray in /anaconda3/lib/python3.6/site-
packages (from NearPy) (0.8.1)
Requirement already satisfied: scipy in /anaconda3/lib/python3.6/site-pac
kages (from NearPy) (1.1.0)
Requirement already satisfied: numpy in /anaconda3/lib/python3.6/site-pac
kages (from NearPy) (1.14.3)
Requirement already satisfied: future in /anaconda3/lib/python3.6/site-pa
ckages (from NearPy) (0.16.0)
You are using pip version 10.0.1, however version 18.0 is available.
You should consider upgrading via the 'pip install --upgrade pip' comman
d.
```

In [2]:
```
import numpy as np
import pandas as pd

from nearpy.distances.distance import Distance
from scipy.stats import entropy
```

Save our BigML Username and Api Key to our environment to access the API

In [3]:
```
import os
os.environ['BIGML_USERNAME'] = "EFETOROS"
os.environ['BIGML_API_KEY'] = "7e5fc6a649fd0f8517fc8ecf2ebd30151c5d4fb4"
```

Creating our main API object with the input of our project id. The project will enable us to organize and keep track of our resources created.

In [4]:
```
from bigml.api import BigML
api = BigML(project="project/5b17b75c92fb560173000387")
```

Downloading Data

For this system, we will be using movie meta data from imdb.

In [5]:
```
movies_df = pd.read_csv("data/movies_metadata.csv",low_memory=False)
```

In [6]:
```
movies_df.head(2)
```

Out[6]:

	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
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	8844	tt0113497

2 rows × 24 columns

We will want to only focus on the title and overview, since we want this workflow to be applied to more than just movies.

In [7]:
```
movies_df = movies_df[["original_title","overview"]]
movies_df.head(2)
```

Out[7]:

	original_title	overview
0	Toy Story	Led by Woody, Andy's toys live happily in his ...
1	Jumanji	When siblings Judy and Peter discover an encha...

We will want to export the filtered data to a CSV because BigML creates sources from CSV files.

In [8]:
```
movies_df.to_csv("data/movies_filtered_topic_model.csv")
```

Importing Data to BigML

In order to start a BigML workflow, a source object has to be created. The API function that creates a source is `create_source` . The method's inputs will be a file path to the csv it will be converting. The source will be created from the csv files written by `to_csv` from before.

In [9]:
```
source = api.create_source("data/movies_filtered.csv")
```

BigML's `ok` method is called in order to assure that an object is created and will wait if it is not done being completed.

In [10]:
```
api.ok(source)
```

Out[10]:
```
True
```

Creating a Dataset

BigML will use the newly created source to create datasets which will enable the API to perform many more operations. In order to create a dataset, the API calls the function `create_dataset` . The method will take the source created by the API as an input.

In [11]:
```
dataset = api.create_dataset(source)

api.ok(dataset)
```

Out[11]:
```
True
```

Building a Topic Model

BigML's API allows for the creation of many models. For this dataset, a Topic Model will be used. The BigML API will use the method `create_topic_model` . Many BigML API functions take in a dictionary with many fields as an additional input. These fields allow for much manipulation of the

original function's outcome. For our Topic Model, we will want to set the number of topics to the max number, which is 64. We want to set the number of topics high because we want the movies descriptions to be represented with detail through topics.

In [12]:
```
# step 5: creating a topc model
topic_model = api.create_topic_model(dataset,{"number_of_topics" : 64})

# waiting for the topic model to be finished
api.ok(topic_model)
```

Out[12]:
```
True
```

Locally Storing Our Model

BigML's API allows for the storage of models Locally by using the function `export` . The inputs of the function will include the model id that is retrieved from the model object, as well as a json path, since the models are stored as .json files.

In [13]:
```
topic_model_id = topic_model["object"]["resource"]
api.export(topic_model_id,
           filename="movies_topic_model.json")
```

Out[13]:
```
'movies_topic_model.json'
```

Creating a Batch Topic Distribution

BigML's API allows for the creation of a batch distribution by using the function `create_batch_topic_distribution` . This step is what is at the core of our recommendation system. We will create a topic distribution with our original dataset. Every instance will have a probability of it being obtained in a topic, for all 64 topic that we have chosen to be in our model. In a sense, these probabilities give each movie a unique DNA that will help us in finding other similar movies.

In [14]:
```
batch_topic_distribution = api.create_batch_topic_distribution( \
    topic_model, dataset)
api.ok(batch_topic_distribution)
```

Out[14]:
```
True
```

We will download the batch topic distributions dataset as a CSV.

In [15]:
```
api.download_batch_topic_distribution( \
    batch_topic_distribution, filename=('data/distribution_predictions.csv')
```

Out[15]:
```
'data/distribution_predictions.csv'
```

We will read our new distribution CSV into our notebook.

In [16]:
```
distributions = pd.read_csv('data/distribution_predictions.csv')
```

In [17]:
```
distributions.head(2)
```

Out[17]:

	Topic 00	Topic 01	Topic 02	Topic 03	Topic 04	Topic 05	Topic 06	Topic 07	Topic 08	Topic 09	...	
0	0.01382	0.00617	0.08283	0.00738	0.05878	0.00009	0.00629	0.00009	0.00009	0.00009	...	0.0
1	0.00022	0.00009	0.07588	0.00009	0.02693	0.00009	0.03665	0.07053	0.00009	0.02608	...	0.0

2 rows × 64 columns

We will create a dictionary of names mapped to their distributions for later use.

In [18]:
```
names = movies_df["original_title"].values
labels = {}
for i in np.arange(len(distributions)):
    labels[names[i]] = distributions.values[i]
```

Implementing Locality-Sensitive Hashing

Next, we will create our distance class that NearPy uses for our locality-sensitive hashing. For our distance formula we will create the Jensen–Shannon divergence, as it is a reliable measure of comparing probability distributions. The reason will will be creating a distance class instead of just a function is because of the structure of NearPy.

In [19]:
```
class js(Distance):
    def distance(self,p, q):
        p = np.asarray(p)
        q = np.asarray(q)
       # normalize
        p /= p.sum()
        q /= q.sum()
        m = (p + q) / 2
        return (entropy(p, m) + entropy(q, m)) / 2
```

We will now create the space that will store of our data, and as stated before we will be using Locality-sensitive hashing implemented through NearPy. We will set two important variables, the dimension of our space, and the number of bits for our hash. We will set the dimension to how many topics we have, 64, and we will want to set the bits to a low number since our dataset is not very big. If we deal with a very large dataset, then we can set a higher number of bits and the performance of getting a reccomendation we still be great.

In [20]:
```
from nearpy import Engine
from nearpy.hashes import RandomBinaryProjections

# Dimension of our vector space
dimension = 64
bits_of_random_binary_hash = 3
```

After we set our two variables, we will create a the random binary hash, as well as an Engine with pipeline configuration from NearPy, that will be organizing our storage space. The engine will also take in our distance formula as an input.

In [21]:
```
rbp = RandomBinaryProjections('rbp', bits_of_random_binary_hash)

engine = Engine(dimension, lshashes=[rbp],distance=js())
```

Next, we will want to give each row the name of the instance from before. Every row gets the title that relates to it, and this labeling is stored in a dictionary for fast retrieval. Then we will add each movies topic distribution and their label into our engine as vectors.

In [22]:
```
names = movies_df["original_title"].values
```

In [23]:
```
for i in np.arange(len(distributions.values)):
    v = distributions.values[i]
    engine.store_vector(v, names[i])
```

We will retrieve our local topic model for faster predictions if a movie is not in our database.

In [24]:
```
import sys
!{sys.executable} -m pip install pystemmer

from bigml.topicmodel import TopicModel
local_topic_model = TopicModel(topic_model_id)
```

```
Requirement already satisfied: pystemmer in /anaconda3/lib/python3.6/site
-packages (1.3.0)
You are using pip version 10.0.1, however version 18.0 is available.
You should consider upgrading via the 'pip install --upgrade pip' comman
d.
```

We create a helper function that will be in our movie reccomemender function. Given a movie name and description, this function will get a topic distribution for the new movie from our local model, and add it into our database.

In [25]:
```
def add_movie(movie_name, description):
    x=local_topic_model.distribution({"original_title": movie_name,
                                      "description": description})
    new_distr = np.array([])
    for item in x:
        new_distr = np.append(new_distr, item["probability"])
    engine.store_vector(new_distr, movie_name)
    labels[movie_name] = new_distr
```

Main Recommender Function

This is our main function that will generate movie reccomendations.

In [26]:
```
def movie_recommender(movie_name):
    #checks if movie_name is in the database
    if movie_name in labels:

        #if the movie_name is valid we use the engine.neighbors function to
        query = np.array(labels[movie_name])
        ten_movies = engine.neighbours(query)
        for ar in ten_movies:
            print(ar[1])
    else:

        #if the movie_name is not valid the function will ask the user to i
        b = input("Sorry, that movie is not in the Database, please input th
        print()
        #after retreiving the input we will add the new movie to the data ba
        add_movie(movie_name, b)
        print()
        #after adding the new movie we will call the movie_recommender funct
        movie_recommender(movie_name)
```

If the movie inputted exists in our database, then our function will output 10 closest neighbors in our defined space. The 10 neighbors will be the 10 movies that relate to each other through their descriptions. The top movie will be the original inputted movie.

In [27]:
```
movie_recommender("Superman")
```

```
Superman
मिस्टर इंडिया
Turbo Kid
Atom Man vs Superman
Batman Unlimited: Mechs vs. Mutants
RoboCop 3
Officer Downe
Justice League: The New Frontier
Dexter's Laboratory: Ego Trip
ドラゴンボールz たったひとりの最終決戦～フリーザに挑んだz戦士 孫悟空の父～
```

If the movie does not exist in our database, the function will ask for for the description, add it to the database, and will out put 10 recommendations.

In [28]:
```
movie_recommender("The Scary Movie")
```

```
Sorry, that movie is not in the Database, please input the description: A
movie about a serial killer who comes to a town.


The Scary Movie
The Flesh and Blood Show
Bram Stoker's Dracula
Devil Dog: The Hound of Hell
Children Shouldn't Play with Dead Things
Ghost
The Unholy
パラノーマル・アクティビティ 第2章 TOKYO NIGHT
Dèmoni
Weirdsville
```

All BigML operations are drawn from the BigML API, and full documentation can be found at BigML API Documentation (https://bigml.com/api). This notebook used the API's python bindings, and full documentation can be found at BigML API Python Bindings (https://bigml.readthedocs.io/en/latest/).

In [ ]:

In [ ]: