A Recommendation System Using BigML's Cluster Analysis

Description

In this notebook, we will build a recommendation system using BigML's Clustering, more specifically, we will take advantage of the batch centroid predictions and the distances the clustering provides.

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/).

In [1]:
```
import numpy as np
import pandas as pd
```

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

In [2]:
```
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 [3]:
```
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 [4]:
```
movies_df = pd.read_csv("data/movies_metadata.csv", low_memory=False)
```

In [5]:
```
movies_df.columns
```

Out[5]:
```
Index(['adult', 'belongs_to_collection', 'budget', 'genres', 'homepage',
'id',
       'imdb_id', 'original_language', 'original_title', 'overview',
       'popularity', 'poster_path', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline', 'title', 'video',
       'vote_average', 'vote_count'],
      dtype='object')
```

We will only focus on the fields of original_title, overview, adult, and the most important feature, its genre.

In [6]:
```
movies_df = movies_df[["original_title","overview","adult","genres"]]
movies_df.head(2)
```

Out[6]:

	original_title	overview	adult	genres
0	Toy Story	Led by Woody, Andy's toys live happily in his ...	False	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...
1	Jumanji	When siblings Judy and Peter discover an encha...	False	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...

In [ ]:
```

```

In [7]:
```
import ast
genres = movies_df["genres"].apply(lambda x: ' '.join(map(str,([y["name"] fo
movies_df["genres"] = genres
```

In [8]:
```
movies_df.head(2)
```

Out[8]:

	original_title	overview	adult	genres
0	Toy Story	Led by Woody, Andy's toys live happily in his ...	False	Animation Comedy Family
1	Jumanji	When siblings Judy and Peter discover an encha...	False	Adventure Fantasy Family

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

In [9]:
```
movies_df.to_csv("data/movies_filtered_cluster.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 [10]:
```
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 [11]:
```
api.ok(source)
```

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

Creating a Dataset

BigML will use the newly created sources 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 sources created by the API as an input.

In [12]:
```
dataset = api.create_dataset(source)

api.ok(dataset)
```

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

Building a Cluster Analysis

BigML's API allows for the creation of many models. For this dataset, clustering will be used. The BigML API will use the method `create_clusterl`. 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 cluster analysis, we will want to set the number of clusters to 100 since we want a good number of clusters that vary. We also want to scale genre (field 000005), since we want it to carry more importance.

In [13]:
```
cluster = api.create_cluster(dataset, {"name": "Movie Clusters",
                                        "k": 100,
                                        "field_scales": {"000002": 1, "00000
                                        "excluded_fields":["000000"],
                                       "default_numeric_value":"zero"})
api.ok(cluster)
```

Out[13]:
```
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 [14]:
```
api.export(cluster,
           filename="movies_cluster.json")
```

Out[14]:
```
'movies_cluster.json'
```

Creating a Batch Centroid

BigML's API allows for the creation of a batch distribution by using the function `create_batch_centroid` . This step is what is at the core of our recommendation system. We will create a batch centroid with our original dataset. Every instance will have a a label for which centroid/cluster it belongs to as well as the distance of that instance to the center of the centroid/cluster. These distances will help us in comparing movies that are similar.

```
In [15]: batch_centroid = api.create_batch_centroid(cluster, dataset, {
             "name": "my_batch_centroid", "all_fields": True,
             "header": True,
             "distance": True,
             "distance_name": "Distance"})
         api.ok(batch_centroid)
```

```
Out[15]: True
```

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

```
In [16]: api.download_batch_centroid( \
             batch_centroid, filename=('data/batch_centroid.csv'))
```

```
Out[16]: 'data/batch_centroid.csv'
```

We will read our new distribution CSV into our notebook.

```
In [17]: batch_df = pd.read_csv('data/batch_centroid.csv')
```

```
In [18]: batch_df.head(2)
```

Out[18]:

	field1	original_title	overview	adult	genres	cluster	Distance
0	0	Toy Story	Led by Woody, Andy's toys live happily in his ...	False	Animation Comedy Family	Cluster 61	0.26542
1	1	Jumanji	When siblings Judy and Peter discover an encha...	False	Adventure Fantasy Family	Cluster 74	0.30515

We will create a pandas group object on cluster. This helps to organize the data and speeds up the process of finding similar movies.

```
In [19]: clusters= batch_df.groupby("cluster")
```

We can enter any cluster and view it as a dataset.

In [20]:
```
clusters.get_group("Cluster 01").head(2)
```

Out[20]:

	field1	original_title	overview	adult	genres	cluster	Distance
1366	1366	The Portrait of a Lady	Ms. Isabel Archer isn't afraid to challenge so...	False	Drama Romance	Cluster 01	0.21777
1696	1696	Love and Death on Long Island	Giles De'Ath is a widower who doesn't like any...	False	Drama Romance Foreign	Cluster 01	0.24154

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

In [21]:
```
from bigml.cluster import Cluster
cluster_id = cluster["object"]["resource"]
local_cluster = Cluster(cluster_id)
```

Main Recommender Function

This is our main function that will generate movie reccomendations.

In [26]:
```
def movie_recommender(movie_name):
    #retrieve the row of the movie we are recommending for
    row = batch_df.loc[batch_df['original_title'] == movie_name]
    row.index= np.arange(len(row))

    #check if the movie_name is in our clusters
    if len(row) >= 1:
        single_cluster = clusters.get_group(row['cluster'][0]).sort_values('
    else:
        #if the movie is not in our clusters, the function will as the user
        #input the overview, a boolean for if the movie is rated R, and most
        #importantly the genres
        print("Sorry that movie is not in the Database")

        overview = input("Please enter the movie overview: ")
        adult = input("Please enter a boolean of if the movie is a rated R:
        genres = input("Please enter a the genre/genres of the movie: ")

        #The function creates a temporary centroid for the new movie
        data = local_cluster.centroid({
        "original_title":movie_name,
        "overview": overview,
        "adult": adult,
        "genres": genres})

        #data will contain centroid_name, which is the cluster name that the
        #movie would live in, as well as the distance to the middle of that
        single_cluster = clusters.get_group(data["centroid_name"])

        #add the new movie as a row into the cluster
        new_row = {"original_title" : movie_name, "adult":adult ,"overview":
        single_cluster = single_cluster.append(new_row,ignore_index=True)

        single_cluster = single_cluster.sort_values("Distance")


    #reset the indexes of the cluster into an ordered range
    single_cluster.index = np.arange(len(single_cluster))

    #find the index of the movie that we are recommending for
    index = single_cluster.loc[single_cluster['original_title'] == movie_nam

    #There are three if statements that account for if a movie
    #is at the end, beginning, or anywhere else of a list of distances.
    #Then retrieve 5 movies that have similar distances to the movie we
    #are recommending for.
    if index < 3:
        return single_cluster.iloc[[index+1,index+2,index+3,index+4,index+5]
    elif index > len(single_cluster) - 4:
        return single_cluster.iloc[[index-1,index-2,index-3,index-4,index-5]
    else:
        return single_cluster.iloc[[index-2,index-1,index+1,index+2,index+3]
```

```
In [23]: movie_recommender("Toy Story")

Out[23]: 50                               Hoodwinked!
         51                          Kronk's New Groove
         53                        Mr. Bug Goes to Town
         54                  The Great Piggy Bank Robbery
         55        Lilo & Stitch 2: Stitch has a Glitch
         Name: original_title, dtype: object
```

```
In [28]: movie_recommender("Superman")

Out[28]: 216                            Solomon Kane
         217                               宇宙からのメッセージ
         219       Dr. Horrible's Sing-Along Blog
         220     The Trial of the Incredible Hulk
         221         Atlantis, the Lost Continent
         Name: original_title, dtype: object
```

```
In [27]: movie_recommender("The Spanish Warrior")

         Sorry that movie is not in the Database
         Please enter the movie overview: A movie about a warrior that saves spain
         from disaster
         Please enter a boolean of if the movie is a rated R: False
         Please enter a the genre/genres of the movie: Action

Out[27]: 357                  あしたのジョー
         358              The Tracker
         360                  Destiny
         361                     龍騰虎躍
         362       American Kickboxer
         Name: original_title, dtype: object
```

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 [ ]:
```