

# INFO7374 Algorithmic Digital Marketing

---

Summary	This app will help movie investors make wiser decisions before they spend money, by predicting the success of the movie.
Author	Yifu Liu and Pengfei He
App URL	<a href="https://quiet-reef-82062.herokuapp.com/">https://quiet-reef-82062.herokuapp.com/</a>

## Objectives

### Target User:

Movie investors.

### Goal

We want to help movie investors make wiser decisions before they spend money. When the investors put some properties of the movie they proposed, like genres, budgets, product companies.

The app will:

- 1 Return similar movies released before as references
- 2 Predict whether the movie will be successful or not.

The definition of success is **box is 1.5 times larger than budget**.

## TMDB Dataset and Preprocessing

### Dataset:

We used TMDB API and the scraping (open source python script) to get the TMDB dataset (we decided not to use IMDB because IMDB dataset recently has some restriction on the rights to use). Rows number: **1934374**

<https://drive.google.com/file/d/1mnH9UaaXZ-gP3At0Q2Qus-Gz0EJOqiYx/view?usp=sharing>  
[https://drive.google.com/file/d/19C8m6CwRu9l-eydnTbp4gPCp\\_eeFMb7t/view?usp=sharing](https://drive.google.com/file/d/19C8m6CwRu9l-eydnTbp4gPCp_eeFMb7t/view?usp=sharing)

```

def make_request(call_url, prior_attempts=0):
    if prior_attempts >= MAX_ATTEMPTS:
        return None
    response = requests.get(call_url)
    if was_rate_limited(response):
        sleep(RATE_LIMITER_DELAY_SECONDS)
    sleep(1 / MAX_DOWNLOADS_PER_SECOND)
    if was_successful(response):
        return response.json()
    else:
        sleep(1) # attempt to sleep through any intermittent issues
        return make_request(call_url, prior_attempts + 1)

def make_detail_request(category, entry_id):
    category_specifics = ''
    if category in CATEGORY_SPECIFIC_CALLS:
        category_specifics = CATEGORY_SPECIFIC_CALLS[category]
    call_url = BASE_API_CALL.format(
        category=category,
        entry_id=entry_id,
        api_key=API_KEY,
        category_specifics=category_specifics,
    )
    return make_request(call_url)

```

## Preprocessing:

Data preprocessing includes drop duplicated values, drop null values, and drop meaningless columns:

```

[ ] # Drop Duplication Values
    duplicated_values = ('num_voted_users', 'popularity', 'budget', 'genres', 'id', 'plot_keywords', 'language', 'original_title', 'overview',
    original_format.drop_duplicates(subset=duplicated_values, keep='first', inplace=True)

[ ] # Homepage and tagline are useless in our model
    original_format.drop(['homepage', 'tagline'], axis=1, inplace=True)

[ ] # Drop rows with invalid voting score
    original_format = original_format[original_format['vote_average'] > 0]

[ ] # Drop rows with invalid duration
    original_format = original_format[10 < original_format['duration']]
    original_format = original_format[original_format['duration'] < 300]

[ ] # Drop rows with invalid budget
    original_format.budget=original_format.budget.astype(int)
    original_format = original_format[original_format.budget!=0]

```

Categorize columns:

Based on the discussion with the Professor in class, we decide to numeralize and give a threshold for each column, so that we can make the similar search in a more clear way. For example, we only allow top ten producing companies in the dataset, all other companies will fall in 'other company'. Thus, when we ask users to put into the company name, they are only allowed to select from these 10 + 1 names. And the similarity search will then become very clear, since there are very clear 11 categories to choose from.

Dataset before:

original_format.tail(8)																		
budget	genres	homepage	id	plot_keywords	language	original_title	overview	popularity	production_companies	production_countries	release_date	box_office	duration	spoken_languages	status	tagline	movie_title	vote_average
1934367	0	Action/Animation	NaN	74816	martial_artsstreet_fighterbased on video game	日本語	スーパーストリートファイターIV	Chun-Li, Guile and Cammy must save earth. Jul...	1.121	[[{"id": 1077, "logo_path": "/1tpNqC8Bpe13uF2...", "name": "JP", "name": "Japan"}]]	2010-04-27	0.0	35.0	[[{"iso_639_1": "ja", "name": "日本語"}]]	Released	NaN	Super Street Fighter IV	6.8
1934368	0	Animation/Action/Adventure	NaN	74817		日本語	マジンガーZ対暗黒大將軍	Kojiro and his friends have defeated Dr. Hell &...	1.638	[[{"id": 5542, "logo_path": "/ayE4LqpAWotavo7x...", "name": "JP", "name": "Japan"}]]	1974-07-25	0.0	43.0	[[{"iso_639_1": "ja", "name": "日本語"}]]	Released	Deep beneath the earth, seven robots lay dormant...	Mazinger Z vs The Great Dark General	5.6
1934369	0	Horror	NaN	74818		English	Blitzkrieg: Escape from Stalag 69	Enter Stalag 69, where torture is just the beginning...	1.723	[[{"id": 3166, "1": "US", "name": "United States of America"}]]	2008-03-01	0.0	120.0	[[{"iso_639_1": "en", "name": "English"}]]	Released	Stalag 69: Where Nazis Rule With Permeation And...	Blitzkrieg: Escape from Stalag 69	5.0
1934370	0	Animation/Adventure/Action	NaN	74819	anime	日本語	マジンガーZ対デビルマン	Mazinger Z vs. Devilman is a 1973 animated movie...	1.575	[[{"id": 5542, "logo_path": "/ayE4LqpAWotavo7x...", "name": "JP", "name": "Japan"}]]	1973-07-18	0.0	43.0	[[{"iso_639_1": "ja", "name": "日本語"}]]	Released	NaN	Mazinger Z vs. Devilman	4.8
1934371	0	Action/Thriller	NaN	74820		English	Hollywood Files	While on a road trip, a man and his sister pick...	1.684	[[{"id": 1069, "logo_path": "/None", "name": "US", "name": "Canada"}, {"id": 3166, "1": "CA", "name": "Canada"}, {"iso_639_1": "en", "name": "English"}, {"iso_639_1": "fr", "name": "French"}]]	2005-06-28	0.0	95.0	[[{"iso_639_1": "en", "name": "English"}, {"iso_639_1": "fr", "name": "French"}]]	Released	NaN	Hollywood Files	5.3
1934372	0	Animation/Action/Adventure/Science Fiction	NaN	74821	super robotgo nagai	日本語	グレートマジンガー対ゲッターロボG 空中大激突	When an UFO delivers a metal-eating monster to...	1.018	[[{"id": 5542, "logo_path": "/ayE4LqpAWotavo7x...", "name": "JP", "name": "Japan"}]]	1975-04-21	0.0	30.0	[[{"iso_639_1": "ja", "name": "日本語"}]]	Released	NaN	Great Mazinger vs. Getter Robo	5.1
1934373	0	Animation/Action/Science Fiction	NaN	74822	ufoliammel/super robotgo nagai	日本語	グレートマジンガー対ゲッターロボG 空中大激突	When a UFO arrives from space and attacks the...	1.84	[[{"id": 5542, "logo_path": "/ayE4LqpAWotavo7x...", "name": "JP", "name": "Japan"}]]	1975-07-26	0.0	25.0	[[{"iso_639_1": "ja", "name": "日本語"}]]	Released	NaN	Great Mazinger vs. Getter Robo G: The Great Space Invaders Invasion	5.6
1934374	0	Action/Adventure/Animation	NaN	74823	giant robotgiant monsterkaiju	日本語	グレンダイザー - 決戦! 大海獣	A giant sea monster known as Dragonaut has...	1.072	[[{"id": 12568, "logo_path": "/None", "name": "JP", "name": "Japan"}, {"id": 3166, "1": "JP", "name": "Japan"}]]	1976-07-18	0.0	31.0	[[{"iso_639_1": "ja", "name": "日本語"}]]	Released	NaN	Grendizer, Getter Robo G, Great Mazinger: Decisive Battle!	6.3

Dataset after:

original_format.head(1)																																	
War	Music	Western	Mystery	History	Horror	TV Movie	Steven Spielberg	Clint Eastwood	Woody Allen	Martin Scorsese	Ridley Scott	Brian De Palma	Steven Soderbergh	Wes Craven	Francis Ford Coppola	Ron Howard	Other Director	Universal Pictures	Paramount	Columbia Pictures	Warner Bros. Pictures	20th Century Fox	New Line Cinema	Walt Disney Pictures	Touchstone Pictures	Miramax	Metro- Goldwyn- Mayer	Other Company					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1					

Budget as example:

```
[ ] # For budget column
original_format.budget=original_format.budget.astype(int)

[ ] # Create threshold to decide five levels of budget: VeryLowBudget, LowBudget, MedBudget, HighBudget, VeryHighBudget
# After this step, each new movie's budget will only fall in one of them. Thus, we can have a much clear way to make
# similar search.
level_1 = original_format.budget[original_format.budget>0].quantile(0.25)
level_2 = original_format.budget[original_format.budget>0].quantile(0.5)
level_3 = original_format.budget[original_format.budget>0].quantile(0.75)
level_4 = original_format.budget[original_format.budget>0].quantile(0.95)

[ ]
original_format['VeryLowBudget'] = original_format['budget'].map(lambda s: 1 if 0< s < level_1 else 0)
original_format['LowBudget'] = original_format['budget'].map(lambda s: 1 if level_1 <= s < level_2 else 0)
original_format['MedBudget'] = original_format['budget'].map(lambda s: 1 if level_2 <= s < level_3 else 0)
original_format['HighBudget'] = original_format['budget'].map(lambda s: 1 if level_3 <= s < level_4 else 0)
original_format['VeryHighBudget'] = original_format['budget'].map(lambda s: 1 if s >= level_4 else 0)

[ ] # Similarly, we also separate the length of movie into three levels
original_format['ShortMovie'] = original_format['duration'].map(lambda s: 1 if s < 90 else 0)
original_format['NormalMovie'] = original_format['duration'].map(lambda s: 1 if 90 <= s < 120 else 0)
original_format['LongMovie'] = original_format['duration'].map(lambda s: 1 if s >= 120 else 0)
```

## Algorithm learned from class: Decision Tree

The algorithm we use to make predictions is the decision tree. We used the default model for sklearn. The accuracy score we get is **0.7**.

### 6 Algorithm learned from Class: Decision Tree

```
▶ # Split data into training and testing data set
y = original_format['success']
y = np.array(y).reshape(-1,1)
x = original_format.drop('success',axis=1)

[50] x_train_all, x_test, y_train_all, y_test = train_test_split(x,y,random_state=10, test_size=.15)
x_train, x_valid, y_train, y_valid = train_test_split(x_train_all, y_train_all, random_state=10, test_size=0.15)

[51] decision_tree = DecisionTreeClassifier(criterion='entropy',max_depth=20, min_samples_leaf=10)
decision_tree.fit(x_train, y_train)

[ ] DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
max_depth=20, max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=10, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort='deprecated',
random_state=None, splitter='best')

[52] score_all=cross_val_score(decision_tree, x_train_all, y_train_all, cv=5)
avg_score=score_all.mean()
print("Accuracy Score based on 5-fold Cross Validation: {}".format(round(avg_score,2)))

[ ] Accuracy Score based on 5-fold Cross Validation: 0.7
```

# Being a Wise Movie Investor App

The application is deployed on Heroku:

<https://quiet-reef-82062.herokuapp.com/>



## Being a Wise Movie Investor

Select the properties of the proposed movie

Select the budget range

Select from the dropdown list

Select the movie length range

Select from the dropdown list

Select the movie genre

Select from the dropdown list

Select the movie director

Select from the dropdown list

Select the produce company

Select from the dropdown list

Made with Streamlit



## Being a Wise Movie Investor

Select the properties of the proposed movie

Select the budget range

LowBudget

Select the movie length range

NormalMovie

Select the movie genre

Action

Select the movie director

Other Director

Select the produce company

Columbia Pictures

Similar movies are as follows:

	original_title	popularity	success	release_year
1385	Desperado	10.4120	no	1995
4590	Silent Rage	5.1230	no	1982
3364	Used Cars	7.5510	yes	1980
3732	Force 10 from Navarone	8.3320	yes	1978
2827	Sinbad and the Eye of the Tiger	8.6730	no	1977

The Movie is not recommended to invest.

Made with Streamlit

## Fast API

As we discussed with the professor in the last class. We can only use streamlit for our application. But we want to demonstrate the knowledge we learned from this class as much as possible. So we also create a version of application which integrate streamlit with fastapi.

```

@app.get('/similar/{query}')
async def get_npf(query):
    params = query.split('&')
    budget = params[0]
    length = params[1]
    genre = params[2]
    director = params[3]
    company = params[4]
    df = similar_data.loc[(similar_data[budget] == 1) & (similar_data[length] == 1) &
                           (similar_data[genre] == 1) & (similar_data[director] == 1) & (similar_data[company] == 1)]
    df_sorted = df.sort_values('release_year', ascending=False).head(5)
    df_display = df_sorted[['original_title',
                             'popularity', 'success', 'release_year']]
    df_display['success'] = df_display['success'].map(
        lambda s: 'yes' if s == 0 else 'no')

    return df_display

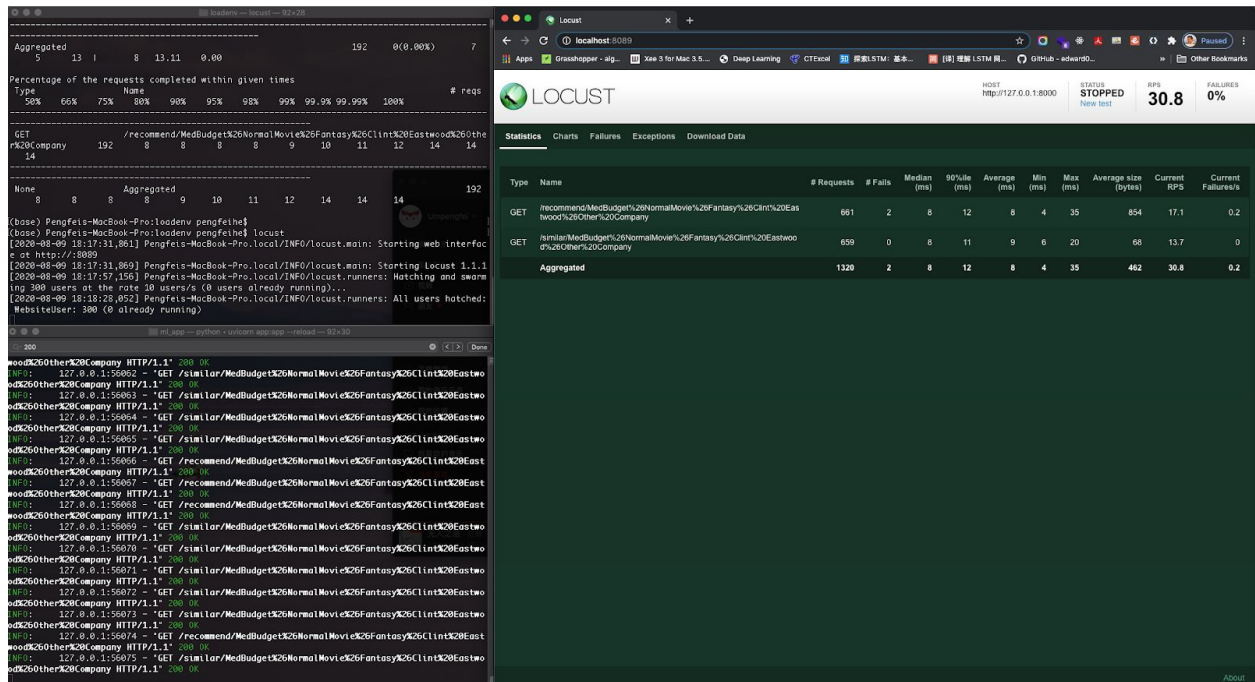
@app.get('/recommend/{query}')
async def get_npa(query):
    params = query.split('&')
    budget = params[0]
    length = params[1]
    genre = params[2]
    director = params[3]
    company = params[4]
    df_pre = predict_data.loc[(similar_data[budget] == 1) & (similar_data[length] == 1) &
                              (similar_data[genre] == 1) & (similar_data[director] == 1) & (similar_data[company] == 1)]
    return df_pre

```

## Load Testing using Locust

We also used locust for a load testing. We tested the FastAPI on the local machine. It is tested on 300 users, 10 user / second, each user will wait 5-15s. The result is as follows:





This api performs quite well. It finished the whole test without making a failure and the response time is very short.

## Special Thank to

Professor: Srikanth Krishnamurthy

TA: Abhishek Maheshwarappa

And everyone in this wonderful summer class!