**Movie Recommender System**

**INFO7390 Advances Data Sci/Architecture**- Final Project Submission

**Aim:** Predicting movies similar to a movie based on plot summary.

**Dataset:** The link to the dataset is :[<https://www.kaggle.com/jrobischon/wikipedia-movie-plots>](https://www.kaggle.com/jrobischon/wikipedia-movie-plots)

**Primary Algorithms:** NLP and K-Means

**Stemming, Tokenisation and creating TfidfVectorizer**

Stemming is the process by which we bring down a word from its different forms to the root word. This helps us establish meaning to different forms of the same words without having to deal with each form separately.

Tokenization is the process by which we break down articles into individual sentences or words, as needed. Besides the tokenization method provided by NLTK, we might have to perform additional filtration to remove tokens which are entirely numeric values or punctuation.

Computers do not understand text. These are machines only capable of understanding numbers and performing numerical computation. Hence, we must convert our textual plot summaries to numbers for the computer to be able to extract meaning from them.

K- Means for Clustering similar movies. (Unsupervised Learning)

TF-IDF recognizes words which are unique and important to any given document

To determine how closely one movie is related to the other by the help of unsupervised learning, we can use clustering techniques. Clustering is the method of grouping together a number of items such that they exhibit similar properties. According to the measure of similarity desired, a given sample of items can have one or more clusters**.**

A good basis of clustering in our dataset could be the genre of the movies. K-means is an algorithm which helps us to implement clustering in Python.

**Technical Implementation**

## Flask

The app.py file contains the main code that will be executed by the Python interpreter to run the Flask web application, it included the ML code for classifying the movies:

* We ran our application as a single module; thus we initialized a new Flask instance with the argument \_\_name\_\_ to let Flask know that it can find the HTML template folder (templates) in the same directory where it is located.
* Next, we used the route decorator (@app.route('/')) to specify the URL that should trigger the execution of the home function.
* Our home function simply rendered the home.html HTML file, which is located in the templates folder.
* Inside the predict function, we access the spam data set, pre-process the text, and make predictions, then store the model. We access the new message entered by the user and use our model to make a prediction for its label.
* we used the POST method to transport the form data to the server in the message body. Finally, by setting the debug=True argument inside the app.run method, we further activated Flask's debugger.
* Lastly, we used the run function to only run the application on the server when this script is directly executed by the Python interpreter, which we ensured using the if statement with \_\_name\_\_ == '\_\_main\_\_'.

Requirements.txt- Contains list of all the libraries to be installed

## HTML

The sub-directory templates is the directory in which Flask will look for static HTML files for rendering in a web browser, in our case, we have one html files: index1.html

The Predict button will run the ML algorithm that we created in the code

Python Code Analysis

1. EDA
2. NLTK Techniques
3. K Means Clustering

Pickle the file- for serializing

Flask- Python

Deploy- Heroku

The Jupyter notebook called- has the respective description for each Comment

**Conclusion**

Training a machine learning model for basics tasks in NLP is simple.

First, data is tokenized and filtered so that it can be represented with units called tokens. We can also represent the words to their root form, so the vocabulary can also be reduced and then we vectorize our dataset using an algorithm that depends on the problem we are trying to solve.

Second, here we used K-Means (Unsupervised Learning) to group similar movies.

Finally, we had a function which took input as a movie and based on the above algorithms/score (similarity distance), it gave us the most similar movie.

This has also been deployed as a Web Application. ( [Web Application](https://moviereco.herokuapp.com/))

Some Sample Inputs that can be used are:

We have supplied 5000 movies- for now these can be used- The Great Train Robbery, Laughing Gas, Batman and so on. Other movies can be chosen from the 5000-movie dataset.

The source code is available on Github and it can be found on the webpage.

**References**

[1] <https://www.oreilly.com/learning/how-do-i-compare-document-similarity-using-python>

[2] <https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50>

Authors:

Praful Praveen Badami

Sheetal Sekhar