Artificial Intelligence and Machine Learning Course Project Report

On

“Multi-Modal Genre Classification for Movies”

Submitted by:

Sahil Singhavi [2019BtechCSE083]

Navneet Gupta[2019BtechCSE075]

Abhay Bhardwaj[2019BtechCSE014]

Submitted to:

Dr. Alok Kumar

Department of Computer Science and Engineering
Institute of Engineering and Technology (IET)
JK Lakshmipat University Jaipur

November 2021

Abstract

Using Machine learning models and algorithms to predict the genres of a particular movie. In this paper, we addressed the multi-label classification of the movie genres in a multimodal way.

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn gradually improving its accuracy. Some real-world applications of machine learning are speech recognition, customer service like online chat bots, Alexa, google assistance, recommendation engines on YouTube etc. To elaborate further let's take movies. Humans can identify the genres of movie very easily. We only require bare minimum information such as posters or name of a movie or dialogs etc. to guess the genre of a movie and in majority of cases we get it right. The idea is to be able to the same with the computer using machine learning, but movie genre classification is a challenging task and is gaining attention a lot.

Here we have used two major types of data visual and textual collected from the internet sites. Textual contains the subtitle and for visuals we have movie posters taken from 152,622 movie titles from The Movie Database (TMDb) and IMDB also. The dataset was carefully curated and organized, making it suitable and useful. Non-deep conventional ML models are primarily used here for the textual data to predict the genres of the movie and we have used deep learning models as well the predict the genres of movie using posters. The project discusses the accuracy of both the models used.

Introduction

In today’s world computer science has improved the standards of living significantly. We have smart phones laptops and other electronic gadgets combined with the Internet make our lives easier and efficient. One of the main technologies which makes these gadgets more user friendly and consumer satisfactory is Artificial intelligence (AI) and Machine learning.

1.a   Machine learning:

Machine learning (ML) is a type of artificial intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. Machine learning algorithms use historical data as input to predict new output values. Machine learning is important because it gives enterprises a view of trends in customer behavior and business operational patterns, as well as supports the development of new products. Many of today's leading companies, such as Facebook, Google and Uber, make machine learning a central part of their operations. Machine learning has become a significant competitive differentiator for many companies.

1.b About the project:

Streaming media services have grown steadily over the past decade as a practical and comfortable way of allowing consumers to have access to films, series, documentaries, and so on. Big companies like Netflix, Amazon prime video, YouTube have hugely invested in this area for making it more user friendly and understanding the demands of the user like what type of genres they prefer and what are the best recommendations for a particular user. Etc. thus, for such type of features ML is the answer. It is an attempt to imitate what the human mind does. We can guess the genre of the movie just by looking at the movie title or poster dialogs etc. thus our perception is very high compared a machine. making a computer do all that work is the aim of ML. We need to test the accuracy of various ML models which predict the genre of a particular movie. For this here there are two types of data available one is textual that is subtitles and the other is visual that is posters. We scrap the data according to our requirements after that use the non-deep conventional model for the textual data. To identify the genres from the posters we use deep learning.

Methodology

In ML, task of classification means to use available data to learn *function* which can assign category to data point. For e.g, assign a genre to a movie, like "Romantic Comedy", "Action", "Thriller". Another example could be automatically assigning category to news articles, like "Sports" & "Politics".

Given:

i) data point x_i
ii) set of categories $y_1, y_2 \ldots y_n$ that x_i belong to.

Task:

Predicting correct category y_k for new data point x_k not present in given dataset.

Problem:

don't know how the x and y are related mathematically.

Assumption:

Assuming exists a function f relating x and y i.e., $f(x_i) = y_i$

Approach:

f not known, learn a function g which approximates f.

Important consideration:

$f(x_i) = g(x_i) = y_i$ for all x_i, after that two functions f and g are exactly equal. This wont realistically ever happen, and we' will only be able to approximate true function f using g. means, sometimes the prediction $g(x_i)$ will not be correct. And essentially, our whole goal is to find g which makes really low number of such errors. .

should mention that this is a specific kind of learning problem which we call "Supervised Learning". the idea that g approximates f well for data do not present in our dataset is called "Generalization". It is paramount that our model generalizes, or else all our claims will only be true about data we already have, and our predictions will not be correct.

There are many other kinds, but supervised learning is most popular and well-studied kind.

Multi-Modal Classification

In ML network, Multi-Modal is refer to multiple *kinds* of data. For example, consider a YouTube video. It can be thought to contain 3 different modalities -

i)video modality
ii)audio modality
iii)textual modality

Consider, that I'm keen on characterizing a melody on Tube as pop or rock can utilize any of the over 3 modalities to foresee the class - The video, the actual tune, or the verses. Be that as it may, can anticipate it much better if could utilize every one of the three at the same time. This is the thing that we mean by multi-modular arrangement.

For this project, we will be using visual and textual data to classify movie genres.

Project Outline

iii) **Scraping dataset**
iv) **Pre-processing dataset**
v) **Introduction Deep Learning**
vi) **ML Models for poster**
vii) **ML Models for Textual data**

Literature Review

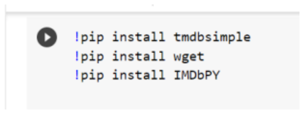
We will scratch information from 2 unique film sources - IMDB and TMDB

IMDB: http://www.imdb.com/

For those uninformed, IMDB is the essential wellspring of data about motion pictures on the web. It is massively rich with banners, audits, rundown, appraisals, and numerous other data on each film. We will utilize this as our essential information source.

TMDB: https://www.themoviedb.org/

TMDB, or The Movie Database, is an open-source rendition of IMDB, with an allowed to utilize API that can be utilized to gather data. do require an API key, however it tends to be gotten free of charge simply by making a solicitation in the wake of making a free record.

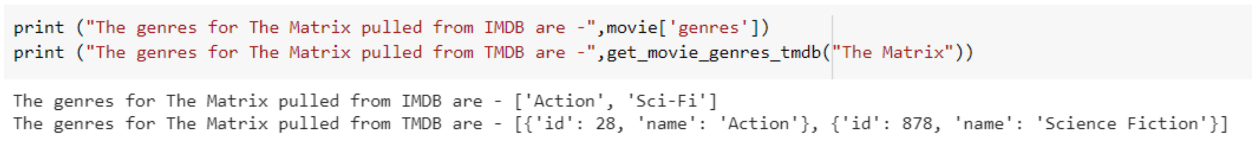


```
!pip install tmdbsimple
!pip install wget
!pip install IMDbPY
```

The data is called from the website mention and how to setup the API and calling our running time data we have described in *[Appendix1.1]*

A comparison of IMDB and TMDB data

Now that we have both systems running, let's do a very short comparison for the same movie.



```
print ("The genres for The Matrix pulled from IMDB are -",movie['genres'])
print ("The genres for The Matrix pulled from TMDB are -",get_movie_genres_tmdb("The Matrix"))
```

```
The genres for The Matrix pulled from IMDB are - ['Action', 'Sci-Fi']
The genres for The Matrix pulled from TMDB are - [{'id': 28, 'name': 'Action'}, {'id': 878, 'name': 'Science Fiction'}]
```

[Appendix1.2]

Building a dataset to work

Utilizing similar API , we will simply pull results from the main 50 pages. As referenced before, the "page" characteristic of the order top movies=all_movies.popular() can be utilized for this reason.

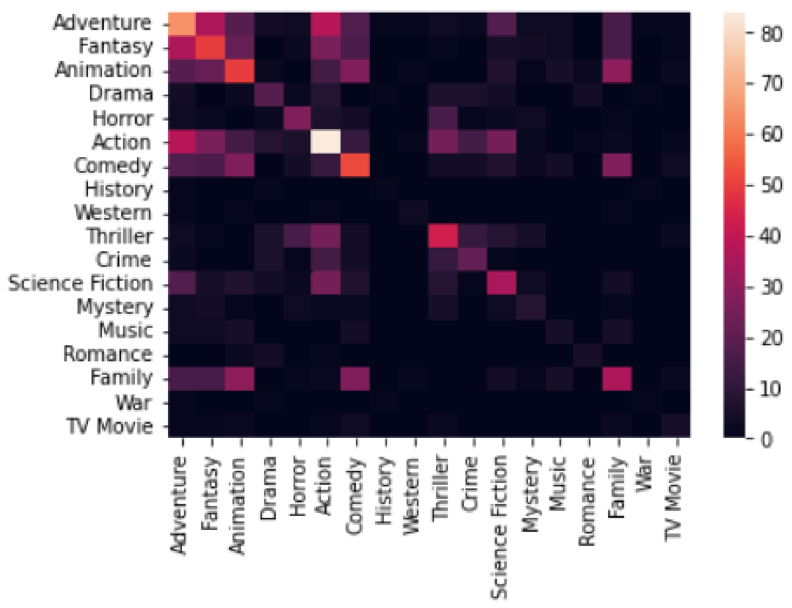
As the data is excessively huge with more properties in a portion of the code underneath will store the data into python "pickle" documents so it tends to be prepared straightforwardly from memory, instead of being downloaded without fail. Once done, should remark out any code which produced an article that was cured and is not generally required.

For reference check the code *[Appendix 2.1]*

Analysis of Movie Genres

Pairwise Analysis

As our dataset is multi label, certainly searching at the distribution of genres is not enough. It might be beneficial to look which genres co-arise, as it'd shed a few light on inherent biases in our dataset.So, for the top one thousand movies allow's do a little pairwise evaluation for genre 'distributions'. Our fundamental purpose is to look which genres arise collectively within the equal movie. So, we first define a characteristic which takes a listing and makes all viable pairs from it. After that, we pull the list of genres for a movie and run this function at the listing of genres to get all pairs of genres which arise together. We made the shape that's a 19X19 shape which shows 19 Genres. This shape counts the wide variety of simultaneous occurrences of genres in same film.



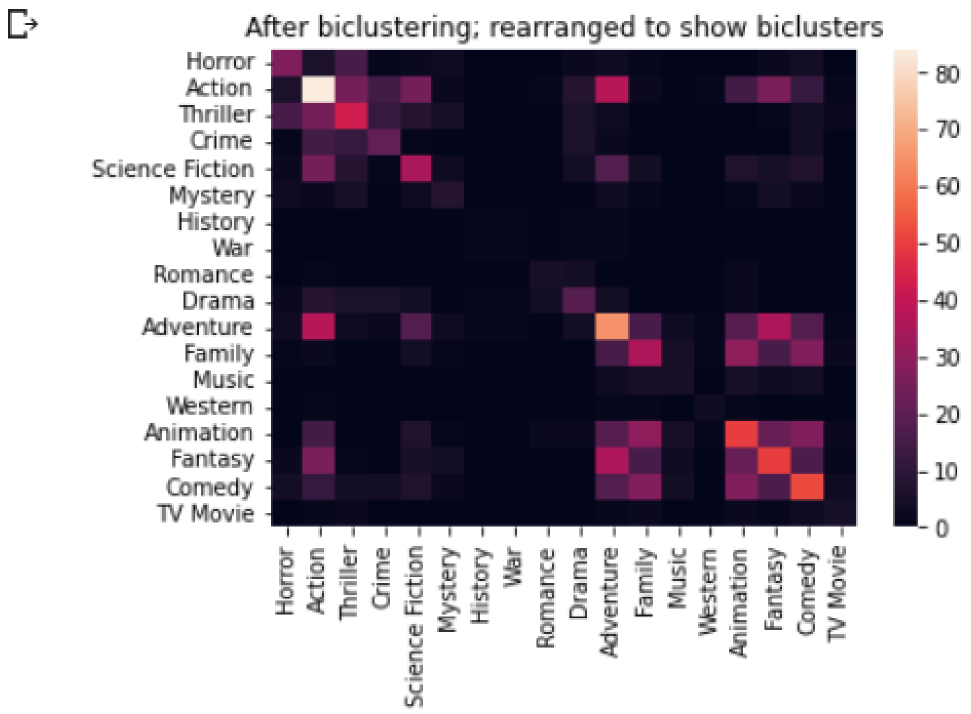
The above image indicates how frequently the genres arise collectively, as a heatmap essential factor to word within the above plot is the diagonal. The diagonal corresponds to self-pairs, i.e., quantity of instances a genre, say Drama took place with Drama. Which is basically just a be counted of the entire instances that style befell! As we are able to see there are a number of dramas within the records set, it is also a very unspecific label. There are nearly no documentaries or tv films. Horror is a distinct label, and romance is also now not too widely unfold.

To account for this unbalanced data, multiple things we can try to explore what interesting relationships can be found.

Delving Deeper into co-occurrence of genres

we want to look for nice groups of genres that co-occur.

while the information might not display that directly, we will play with the numbers to peer if that's feasible. The method used for this is known as biclustering.



searching at the above figure, "boxes" or groups of movie genres automatically emerge! Intuitively - Crime, Sci-Fi, thriller, action, Horror, Drama, thriller, and so forth. co-occur. AND, Romance, myth, circle of relatives, song, adventure, and many others. co-occur.

For code of above check *[Appendex2.4]*

One assignment is the large variety of the drama style. It makes the two clusters exceptionally overlapping. If we merge it together with motion thriller, etc. we are able to emerge as with nearly all films just having that label.

based on playing round with the stuff above, we are able to sort the information into the subsequent style categories - "Drama, movement, technological know-how Fiction, thrilling (thriller, crime, mystery), uplifting (adventure, delusion, animation, comedy, romance, circle of relatives), Horror, datasets"

This categorization is subjective and in no way the simplest proper solution. One can also simply live with the original labels and best exclude the ones with now not enough facts. Such hints are vital to balance the dataset, it allows us to increase or decrease the strength of certain indicators, making it viable to improve our inferences.

thoughts to discover in particular for feature correlations:

- **Is length correlated with film style?**
- **Are film posters darker for horror than for love end comedy?**
- **Are some genres in particular released extra frequently at a sure time of year?**
- **Is the RPG score correlated with the genre?**

Based on this new category set, we will now pull posters from TMDB as our training data!
Check *[Appendix 1.3]*

Scraping done!

Dataset Building from scraped information!

Because the data is scraped now, we want it to apply it however nevertheless in pieces so we want to combine and clean the data. This task is easy, however extraordinarily vital. it's basically what's going to set the degree for the entire project. given that have the freedom to cast their own challenge in the framework supplying, there are numerous decisions that have to make to finalize very own model of the task.

As we are working on a classification problem, we need to make two decisions given the data at hand -

viii) What do we want to predict, i.e., what's our Y?
ix) What features to use for predicting this Y, i.e., what X should we use?

There are many different options possible, and it comes down to decide what's most exciting. I will be picking my own version for the example, but it is imperative that think this through, and come up with a version which excites!

As an example, here are some possible ways to frame Y, while still sticking to the problem of genre prediction -

x) Assume every movie can have multiple genres, and after that it becomes a multi-label classification problem. For example, a movie can be Action, Horror and Adventure simultaneously. Thus, every movie can be more than one genre.

xi) Make clusters of genres as we did in Milestone 1 using biclustering, and after that every movie can have only 1 genre. This way, the problem becomes a simpler, multi-class problem. For example, a movie could have the class - Uplifting (refer Milestone 1), or Horror or History. No movie gets more than one class.

For the purposes of this implementation, I'm going with the first case explained above - i.e., a multi-label classification problem.

Similarly, for designing our input features i.e., X, may pick any features think make sense, for example, the Director of a movie may be a good predictor for genre. OR they may choose any features they design using algorithms like PCA. Given the richness of IMDB, TMDB and alternate sources like Wikipedia, there is a plethora of options available. Another important thing is that in doing so, we must also make many more small implementation decisions on the way. For example, what genres are we going to include? what movies are we going to include? All these are open ended!

As we on a classification probleme, we want to make two choices given the records at hand -

- What do we want to predict, i.e., what's our Y?

- Features to apply for predicting the Y, i.e., what X to use?

there are numerous distinct options feasible, and it comes all the way down to determine what is most thrilling. I can be choosing my own version for the instance, but it is vital that suppose this via, and give you a version which excites!

as an example, right here are a few viable approaches to frame Y, whilst nonetheless sticking to the problem of genre prediction -

• count on every movie may have more than one genres, after which it will become a multi-label classification problem. for instance, a film can be motion, Horror and adventure concurrently. hence, each movie may be a couple of style.

• Make clusters of genres as we did in Milestone 1 the use of biclustering, after which every movie can have most effective 1 genre. This manner, the problem becomes a simpler, multi-class problem. as an example, a movie ought to have the class - Uplifting (refer Milestone 1), or Horror or history. No film gets more than one class.

For the purposes of this implementation, we going with the first case defined above - i.e., a multi-label type problem.

similarly, for designing our enter functions i.e., X, may pick any feature think make sense,

Implementation

Implementation decisions made -

xii) The problem is framed here as a multi-label problem explained above.
xiii) We will try to predict multiple genres associated with a movie. This will be our Y.
xiv) We will use 2 different kinds of X - text and images.
xv) For the text part - Input features being used to predict the genre is a form of the movie's plot available from TMDB using the property 'overview'. This will be our X.
xvi) For the image part - we will use the scraped poster images as our X.

We will first look at some conventional machine learning models, which were popular before the recent rise of neural networks and deep learning. For the poster image to genre prediction, I have avoided using this because conventional ML models are simply not used anymore without using deep learning for feature extraction (all discussed in detail ahead, don't be scared by the jargon). For the movie overview to genre prediction problem, we will look at both conventional models and deep learning models.

let's build our X and Y!

First, let's identify movies that have overviews. Next few steps are going to be a good example on why data cleaning is important!

Now let's store the genres for these movies in a list that we will later transform into a binarized vector.

Binarized vector representation is a very common and important way data is stored/represented in ML. Essentially, it's a way to reduce a categorical variable with n possible values to n binary indicator variables. What does that mean? For example, let [(1,3), (4)] be the list saying that sample A has two labels 1 and 3, and sample B has one label 4. For every sample, for every possible label, the representation is simply 1 if it has that label, and 0 if it doesn't have that label. So, the binarized version of the above list will be –

Implementation decisions made -

1)The problem is framed right here as a multi-label problem explained above.

2)we can try and are predicting multiple genres associated with a film. this could be our Y.

3)we will use 2 exclusive kind of X - text and pics.

4)For the text part - entered features being used to predicting the genre is a shape of the movie's plot available from TMDB using the property 'review'. this will be our X.

- For image part - we can use the scraped poster image as our X.

we are able to first look at some traditional machine learning models, which were famous before the recent upward push of neural networks and deep learning. For the poster photo to style prediction, we have avoided the usage of this due to the fact conventional ML models are clearly not used anymore with out the use of deep learning for feature extraction For the movie overview to genre prediction problem, we can have a look at both conventional and deep learning models.

let's construct our X and Y!

First, let's discover films which have overviews.

Now allow's keep the genres for those films in a listing that we are able to later rework into a binarized vector.

Binarized vector illustration is a common and essential manner data is saved/represented in ML. basically, its wayr to lessen a express variable with n feasible values to n binary indicator variables.

[(1,0,1,0]),
(0,0,0,1])]

See *[Appendix 1.4]*

Storing movie overview in a matrix

The way we will be storing the X matrix is called a "Bag of words" representation. The basic idea of this representation in our context is that we can think of all the distinct words that are possible in the movies' reviews as a distinct object. And after that every movie overview can be thought as a "Bag" containing a bunch of these possible objects.

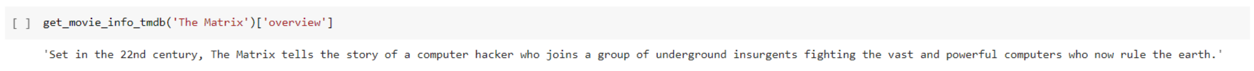
The simple concept of this representation in our context is that we are able to think of all the distinct words which can be viable inside the films' reviews as a awesome object. and after that every movie assessment may be idea as a "Bag" containing a gaggle of these viable objects.

What this means is that, for all the movies that we have the data on, we will first count all the unique words. Say, there's 30,000 unique words. After that we can represent every movie overview as a 30000x1 vector, where each position in the vector corresponds to the presence or absence of a particular word. If the word corresponding to that position is present in the overview, that position will have 1, otherwise it will be 0.

What this indicates is that, for all the movies that we've the statistics on, we will first remember all of the particular phrases. Say, there may be 30,000 particular phrases. After that we will constitute every film overview as a 30000x1 vector, wherein every role in the vector corresponds to the presence or absence of a specific phrase. If the word corresponding to that role is present in overview, that position will have 1, in any other case it will likely be zero.

[*appendix 2.5]*

For example, let's consider the overview for Matrix Mpvie –



```
[ ]  get_movie_info_tmdb('The Matrix')['overview']

     'Set in the 22nd century, The Matrix tells the story of a computer hacker who joins a group of underground insurgents fighting the vast and powerful computers who now rule the earth.'
```

For "The Matrix" a phrase like "computer" is a more potent indicator of it being a Sci-Fi film, than phrases like "who" or "effective" or "huge". One way computer scientists operating with natural language tackled this trouble inside the beyond (and it is still used very popularly) is what we name TF-IDF i.e., time period Frequency, Inverse file Frequency. The fundamental concept here is that phrases which are strongly indicative of the content of a single document (every film assessment is a record in our case) are words that occur very frequently in that document, and really infreuently in all other documents.

For three variables, it'd be one hundred thousand, and we might want to pattern at 500,000 points. that is already extra than the range of information points we've for maximum training problem we are able to ever stumble upon.

essentially, as the dimensionality (number of functions) of the examples grows, due to the fact a hard and fast-size training set covers a dwindling fraction of the input space. even with a mild size of 100 and a large training set of a thousand billion examples, the latter covers handiest a fragment of approximately 10^{-18} of the input area. this is what makes device studying both essential and difficult.

So, yes, if a few words are unimportant, we need to do away with them and reduce the dimensionality of our X matrix. And the manner we will do it is the use of TF-IDF to become aware of un-crucial words. Python shall we us try this with simply one line of code

```
vectorize=CountVectorizer(max_df=0.95, min_df=0.005)
X=vectorize.fit_transform(content)
```

We are excluding all words that occur in too many or too few documents, as these are very unlikely to be discriminative. Words that only occur in one document most probably are names, and words that occur in nearly all documents are probably stop words. Note that the values here were not tuned using a validation set. They are just guessing. It is ok to do because we didn't evaluate the performance of these parameters. In a strict case, for example for a publication, it would be better to tune these as well.

So, each movie's overview gets represented by a 1x1201 dimensional vector.

Now, we are ready for the kill. Our data is cleaned, hypothesis is set (Overview can predict movie genre), and the feature/output vectors are prepped. Let's train some models!

we are apart from all phrases that arise in too many or too few documents, as these are impossible to be discriminative. words that most effective arise in a single file maximum probably are names, and words that arise in almost all documents are stop words. note that the values here were not tuned the usage of a validation set. they're simply guessing. it's miles ok to do due to the fact we didn't evaluate the performance of those parameters. In a strict case, as an example for a booklet, it would be better to tune those as properly.

So, each film's overview gets represented through a 1x1201 dimensional vector.

we're ready for the kill. data is cleaned, hypothesis is about (overview pedicting movie genre), and the feature/output vectors are prepped. let's train!

[Appendix 2.6]

Data set ready

we are building our own dataset, and didn't want to spend all time waiting for poster image downloads to finish, we are working with extremely small dataset. That is why, the results we will will not be spectacular as compared to conventional machine learning methods.

Non-deep, Conventional ML models with above data

Here is what we will be doing -

1,implementing two different models
2,Deciding a performance metric
3. Difference between the models, their strengths, weaknesses, etc.(Discussion)

There are many implementation decisions . Between feature engineering, hyper-parameter tuning, model selection and how interpretable do want model to be (Read: Bayesian vs Non-Bayesian approaches) a lot is to be decided. For example, some of these models could be:

i)Generalized Linear Models
ii)SVM
iii)Shallow (1 Layer, i.e., not deep) Neural Network

iv)Random Forest
v)Boosting
vi)Decision Tree

Or Bayesian:

i)Naive Bayes
ii)Linear or Quadratic Discriminant Analysis
iii)Bayesian Hierarchical models

The Endless lists, and not all models will make sense for the kind of problem have framed for self.

For our purpose , showing the example of 2 very simple models,

i)SVM
ii)Multinomial Naive Bayes

whole pipeline below(Overview):
i)A little bit of feature engineering
ii) different Models
iii)Evaluation Metrics chosen
iv)Model comparisons

Second, can only represent based on the data at hand.

A nice way to think of it is to think that start with the problem at hand, but design features constrained by the data have available. If have many independent features that each correlate well with the class, learning is easy. On the other hand, if the class is a very complex function of the features, may not be able to learn it.

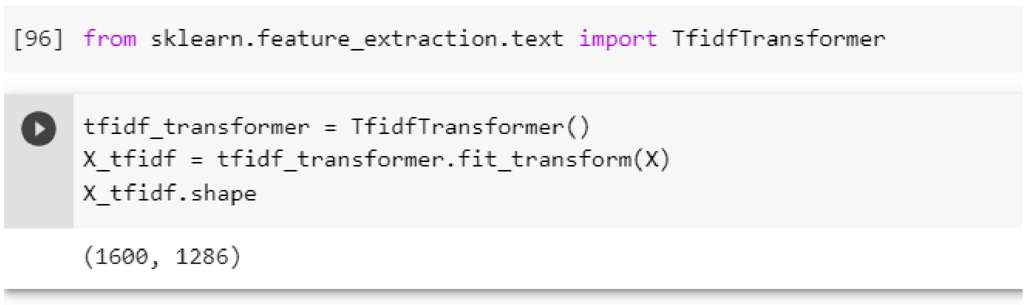
in the context of the problem, we would like to predict genre of a film. what we have get admission to to - movie overviews, which might be textual content descriptions of the film plot. The speculation makes sense, review is a quick description of the story and the story is certainly essential in assigning genres to movies.

So, lets enhance our functions by way of gambling with the words within the overviews in our datas. One interesting way to head returned to what we mentioned earlier - TF-IDF. We originally used it to clear out word, however we can also assign the tied values as "significance" values to words, as opposed to treating all of them equally. Tied truly attempts to become aware of the assign a weightage to every word in the bag of phrases.

yet again, the manner it really works is - most film descriptions have the word "The" in it. obviously, it would not tell whatever unique about it. So, weightage have to be inversely proportional to how many movies have the word of their description. that is the IDF element.

however, for the film interstellar, if the description has the phrase space 5 instances, and wormhole 2 instances, after that it is likely greater about space than about wormhole. thus, space must have a excessive weightage. that is TF element.

We genuinely use TF-If to assign weightage to every word inside the bag of phrases



```
[96] from sklearn.feature_extraction.text import TfidfTransformer
```

```
tfidf_transformer = TfidfTransformer()
X_tfidf = tfidf_transformer.fit_transform(X)
X_tfidf.shape
```

```
(1600, 1286)
```

allow's divide our X and Y matrices into train and test split. Training the model on train split and report the performance at the test slip . consider this like the questions do inside the prolem units v/s the exam. Of direction, they are both (assumed to be) from the same populace of questions. And doing well on problem units is a good indicator that 'do well in tests, however simply, ought to check before can make any claims approximately knowing the subject.

[Appendix 1.5]

inside the output we observe the performance is via and big poorer for movies which can be less represented like battle and animation, and better for classes like Drama.

Numbers apart, permit's have a look at our model predictions for a small pattern of movies from our test set.

As seen above, the outcomes appear promising, however how do we virtually compare the 2 models? We need to quantify our overall performance so that we are able to say which one's better. Takes us again to what we mentioned right within the beginning – we learning the function g which could approximate the unknown function f. For a few values of x_i, the predictions can be wrong for certain, and we need to reduce it.

For multi label structures, we often keep tune of overall performance using "Precision" and "recall". these are popular metrics and may google to study up greater approximately them if there are new to these words.

[Appendix 1.6]

Here is Evaluation Matrix

Using standard precision recall metrics for evaluating systems average precision and recall score for samples are pretty good! Model working!

[appendix1.6]

Deep Learning(Overview)

above outcomes have been excellent, but it's time to deliver out the big guns. So, first and main, permit's get a completely short concept approximately what is deep learning.

As described above, two most crucial principles in doing excellent classification (or regression) are to

1) Use the proper illustration which captures the info about the data which is which is relevant to the problem at hand

2) the use of the right model which has the capability of making sense of the representation fed to it.

To simply emphasize the importance of illustration in the complex tasks we generally try with Deep Learning, let us talk the original problem which made it famous. The paper is frequently called the "ImageNet task Paper", and it was essentially working on object recognization in pictures.

The "Deep" within the deep learning comes from the fact that it turned into conventionally carried out to Neural Networks. Neural Networks, as we all recognise, are structures organized in layers. Layers of computations. Why do we need layers? due to the fact those layers be sub-tasks that we do within the complicated task of recognizing out a chair. it is able to be thought as a hierarchical split down of a complex task into smelled sub-tasks.

Mathematically, each layer acts like space transformation which takes the pixel value to a excessive dimensional space. whilst we start off, every pixel inside the image is given identical importance in our matrix. With each layer, convolution operations deliver a few elements greater importance, and a few lesser significance. In doing so, we rework our images to space in which similar searching objects/object parts are closer

What precisely became learnt by those neural networks is tough to know, and an active vicinity of research. but one very crude way to visualize what it does is to assume like - It starts offevolved by generic features within the first layer. something as simple as vertical and horizontal traces. within the next layer, it learns that if integrate the vectors representing vertical and horizontal vectors in different ratios, can make all feasible slanted strains. subsequent layer learns to combine lines to form curves - Say, some thing like the define of a face. those curves come together to form 3-D objects. and so forth. building sub-modules, combining them within the proper manner that deliver it semantics.

So, in a nutshell, the primary few layers of a "Deep" community examine the proper illustration of the facts, given the problem (that is mathematically defined with the aid of goal characteristic seeking to decrease difference between ground truth and predicted labels). The last layer absolutely seems how near or far apart things are in this high dimensional space.

subsequently, we will supply any sort of data a high dimensional illustration the usage of neural networks. beneath we can see high dimensional representations of each words in overviews (text) and posters (photograph). allow's get commenced with the posters i.e., extracting visual features from posters using deep learning .

Predicting genre from poster

once again, we have to make an implementation decision. This time, it has greater to do with how tons time are we willing to spend in return for brought accuracy. we are going to use here a way this is generally called Pre-Training in machine Learning.

in place of seeking to re-invent the wheel here, going to borrow this short segment on pre-training from Stanford college's lecture on CNN's. to quote -

"In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pretrain a Convent on a very large dataset (e.g., ImageNet, which contains 1.2 million images with 1000 categories), and after that use the Convent either as an initialization or a fixed feature extractor for the task of interest. "

What this indicates, is that within the area spacestack transforms the snap shots to, all photographs which include a "dogse" are closer to each other, and all photos containing a "cat" are closer. consequently, it is a significant area wherein photographs with similar objects are closer.

think about it, now if we pump our posters through this stack, it will embed them in space where posters which include similar objects are closer While a smiling couple could point to romance or drama. The opportunity might be to train the CNN from scratch that is computationally intensive and entails a whole lot of tricks to get the CNN training to converge to the most efficient area transformation.

This manner, we can start off with something robust, after which build on top. We pump our pictures via the pre-trained network to extract the visual features from the posters. After that, the use of those functions as descriptors for the picture, and genres as labels, we train a less difficult neural network from scratch which learns to classification in this dataset. these 2 steps are precisely what we're going to do for predicting genres from film posters..

Extracting visual features from posters

The problem here we answering is can we use poster to predict genre. is hypothesis make sense? Yes. This what graphic designers do g. leaving visual cues to semantics. They make sure that when we look poster of horror movie, we can know it's not smiling image. Can our deep learning system infer such subtleties?

either we train deep neural network ourselves from scratch, or we can use a pre-trained made available to us from Visual Geometry Group at Oxford University, one of the most popular method. called the VGG-net. . , it's just space transformation in form of layers.

Kera's is library that makes it easy for uss. Some other are TensorFlow and PyTorch. While the Kera's makes it easy for prototype by keeping the syntax simple. Some common ways people refer to step are - "Getting the VGG features of an image", or "Forward Propagating the image through VGG and chopping off the last layer".

[Appendix2.7]

Training a simple neural network model using these VGG features.

let's first get labels on 1570 samples As picture download fails on a few instances, the high-quality way to work with the right model is to examine the poster names downloaded and running from there. these posters can't be uploaded to GitHub as they're too large size

[Appendix 1.7]

The final movie poster set for which we have all the information we need, is movies.

In above code we're making an X NumPy array containing the visual feature of 1 picture in step with row. VGG function are reshaped to be in form (1,25088) and we obtain a matrix of shape (1553,25088).

Our binarized Y NumPy array includes binarized labels corresponding to genre IDs of 1553 films.

Now, we create our own koras neural network to use the VGG functions and after that classify movie genres.

Neural network architectures have gotten complicated over the years. however the simplest ones comprise very wellknown computations organized in layers, as described above. Given the popularity of a number of these, Kera's makes it as smooth as writing out the names of those operations in a sequential order. This manner can make networl while completely keeping off Math

Sequential () lets in us to make models the follow this sequential order of layers. unique forms of layers like Dense, Conv2D etc. can be used, and many activation features like RELU, Linear etc. also are to be had.

Let's train our model after that from features extracted from VGG net

Just one hidden layer between the VGG features and the final output layer. The simplest neural network can get. An image goes into this network with the dimensions (1,25088), the first layer's output is 1024 dimensional. This hidden layer output undergoes a pointwise RELU activation. output gets transformed into the output layer of 20 dimensions. It goes through a sigmoid.

The sigmoid, is a function which squashes numbers between 0 and 1.

By squashing score of each 20 output labels between 0 and 1, sigmoid lets us interpret their scores as probabilities. After that we can pick the classes with top 3 or 5 probability scores as predicted genres for movie poster

[Appendix 1.8]

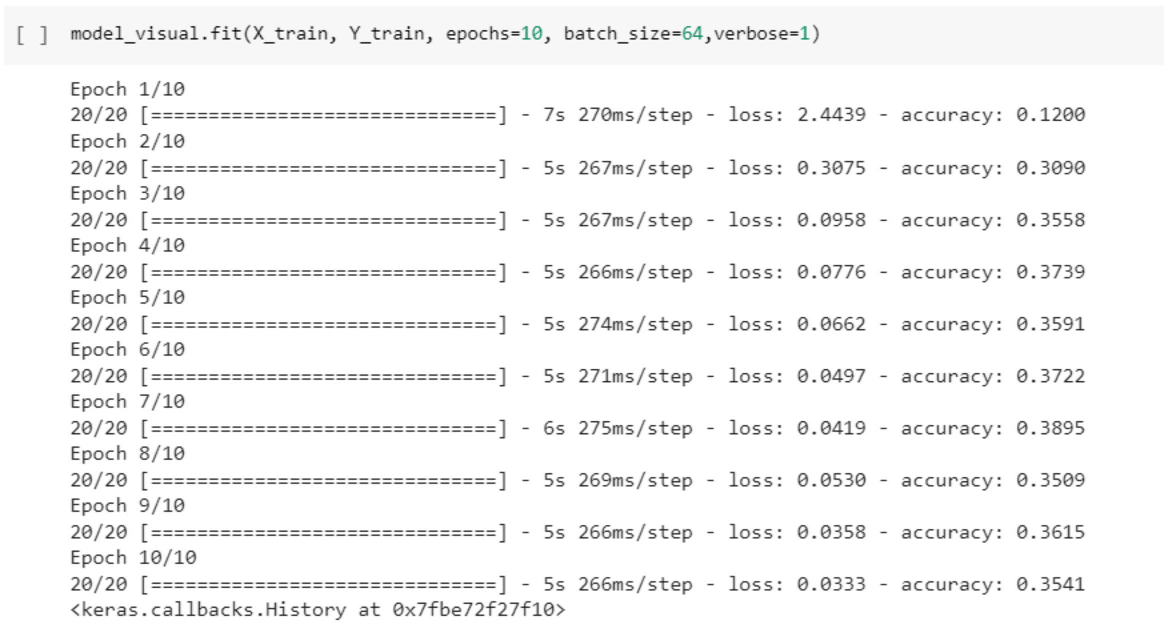
Trained the model using the fit () function. It takes these parameters –

training features and training labels, epochs, batch size and verbose.

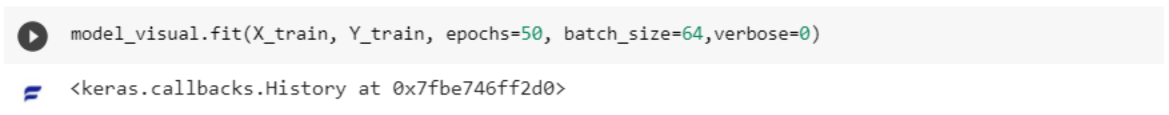
verbose. 0="don't print anything as work", 1="Inform me as go".

data set is too large to be loaded into RAM. we load data in batches. For batch size=32 and epochs=10, model loading starts rows from X in batches of 32 every time it calculates the loss and updates the model.and keeps going until it has covered all the samples 10 times.

no. of times model updated = (Total Samples/Batch Size) x (Epochs)



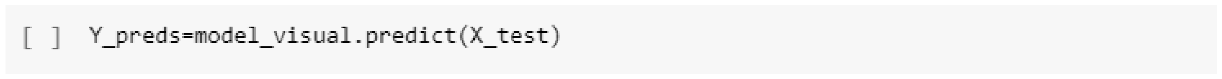
```
[ ] model_visual.fit(X_train, Y_train, epochs=10, batch_size=64,verbose=1)

Epoch 1/10
20/20 [==============================] - 7s 270ms/step - loss: 2.4439 - accuracy: 0.1200
Epoch 2/10
20/20 [==============================] - 5s 267ms/step - loss: 0.3075 - accuracy: 0.3090
Epoch 3/10
20/20 [==============================] - 5s 267ms/step - loss: 0.0958 - accuracy: 0.3558
Epoch 4/10
20/20 [==============================] - 5s 266ms/step - loss: 0.0776 - accuracy: 0.3739
Epoch 5/10
20/20 [==============================] - 5s 274ms/step - loss: 0.0662 - accuracy: 0.3591
Epoch 6/10
20/20 [==============================] - 5s 271ms/step - loss: 0.0497 - accuracy: 0.3722
Epoch 7/10
20/20 [==============================] - 6s 275ms/step - loss: 0.0419 - accuracy: 0.3895
Epoch 8/10
20/20 [==============================] - 5s 269ms/step - loss: 0.0530 - accuracy: 0.3509
Epoch 9/10
20/20 [==============================] - 5s 266ms/step - loss: 0.0358 - accuracy: 0.3615
Epoch 10/10
20/20 [==============================] - 5s 266ms/step - loss: 0.0333 - accuracy: 0.3541
<keras.callbacks.History at 0x7fbe72f27f10>
```



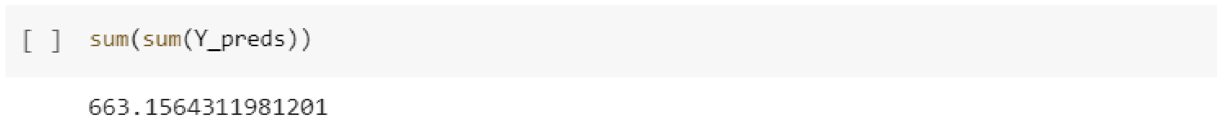
```
model_visual.fit(X_train, Y_train, epochs=50, batch_size=64,verbose=0)

<keras.callbacks.History at 0x7fbe746ff2d0>
```

For top 10 epochs trained model in averbose fashion for telling what happening

After I turned off the verbosity which keeps the code cleaner.



```
[ ] Y_preds=model_visual.predict(X_test)
```



```
[ ] sum(sum(Y_preds))

663.1564311981201
```

Some Predictions

```
f6=open('Genredict.pckl','rb')
Genre_ID_to_name=pickle.load(f6)
f6.close()
```

```
sum(Y_preds[1])
```

```
0.9997553249269232
```

```
sum(Y_preds[2])
```

```
1.9156346032248557
```

```
genre_list=sorted(list(Genre_ID_to_name.keys()))
```

```
precs=[]
recs=[]
for i in range(len(Y_preds)):
    row=Y_preds[i]
    gt_genres=Y_test[i]
    gt_genre_names=[]
    for j in range(19):
        if gt_genres[j]==1:
            gt_genre_names.append(Genre_ID_to_name[genre_list[j]])
    top_3=np.argsort(row)[-3:]

    predicted_genres=[]
    for genre in top_3:
        predicted_genres.append(Genre_ID_to_name[genre_list[genre]])
    (precision,recall)=precision_recall(gt_genre_names,predicted_genres)
    precs.append(precision)
    recs.append(recall)
    if i%50==0:
        print ("Predicted: ",','.join(predicted_genres)," Actual: ",','.join(gt_genre_names))
```

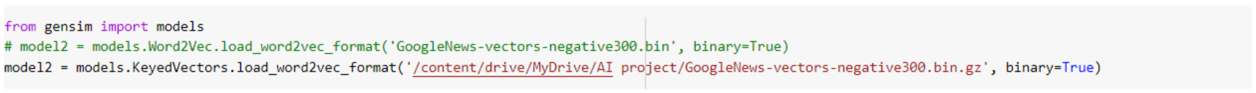
```
Predicted:  Adventure,Comedy,Fantasy  Actual:  Adventure,Fantasy,Action
Predicted:  Animation,Adventure,Action  Actual:  Animation,Action,Science Fiction,Family
Predicted:  Animation,Action,Adventure  Actual:  Adventure,Fantasy,Action
Predicted:  Drama,Action,Thriller  Actual:  Action,Western,Science Fiction
Predicted:  Mystery,Music,Documentary  Actual:  Documentary
Predicted:  Comedy,Crime,Music  Actual:  Documentary,Music
Predicted:  Family,Science Fiction,Comedy  Actual:  Adventure,Comedy,Science Fiction,Family
```

So, even with just the poster ofcourse text outperforms the visual features, but the thing is that it still works. In more complicated models, we can combine the two to make even better predictions.

These models were trained in CPU's, and a simple 1-layer model was used to show becaset there is a lot of information in this data that the models can extract. With a larger dataset, and more training we able to bring these numbers to as high as 70%, same as textual.

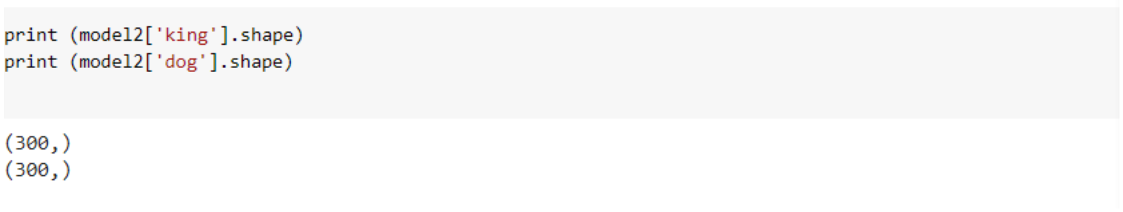
For getting Textual Features

We will use shelf representation for words - Word2Vec model.. As words in total number is small, we even don't need to forward propagate our sample . Even that has been done for us, and the result is stored in the form of a dictionary. simply just look up word in the dictionary and get the Word2Vec features for the word



```
from gensim import models
# model2 = models.Word2Vec.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
model2 = models.KeyedVectors.load_word2vec_format('/content/drive/MyDrive/AI project/GoogleNews-vectors-negative300.bin.gz', binary=True)
```

For eg.

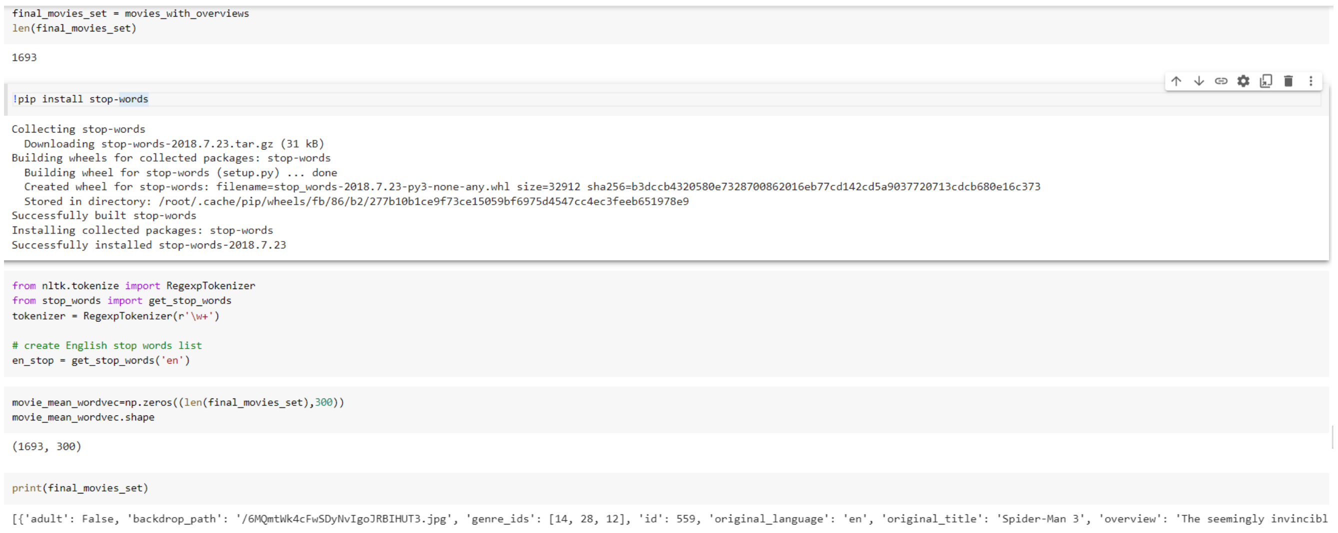


```
print (model2['king'].shape)
print (model2['dog'].shape)
```

```
(300,)
(300,)
```

 Words can be represented in overview using word2vec model. After that, we can use that as our X representations. So, instead of count of words, we are using a representation which is based on the semantic representation of the word. Mathematically, each word went from 3 dimensional length to 300 dimensions!

For the same set of movies above, let's predict t genres from deep representation of overviews!



```
final_movies_set = movies_with_overviews
len(final_movies_set)
```

```
1693
```

```
!pip install stop-words
```

```
Collecting stop-words
  Downloading stop-words-2018.7.23.tar.gz (31 kB)
Building wheels for collected packages: stop-words
  Building wheel for stop-words (setup.py) ... done
  Created wheel for stop-words: filename=stop_words-2018.7.23-py3-none-any.whl size=32912 sha256=b3dccb4320580e7328700862016eb77cd142cd5a9037720713cdcb680e16c373
  Stored in directory: /root/.cache/pip/wheels/fb/86/b2/277b10b1ce9f73ce15059bf6975d4547cc4ec3feeb651978e9
Successfully built stop-words
Installing collected packages: stop-words
Successfully installed stop-words-2018.7.23
```

```
from nltk.tokenize import RegexpTokenizer
from stop_words import get_stop_words
tokenizer = RegexpTokenizer(r'\w+')

# create English stop words list
en_stop = get_stop_words('en')
```

```
movie_mean_wordvec=np.zeros((len(final_movies_set),300))
movie_mean_wordvec.shape
```

```
(1693, 300)
```

```
print(final_movies_set)
```

```
[{'adult': False, 'backdrop_path': '/6MQmtWk4cFwSDyNvIgoJRBIHUT3.jpg', 'genre_ids': [14, 28, 12], 'id': 559, 'original_language': 'en', 'original_title': 'Spider-Man 3', 'overview': 'The seemingly invincibl
```

textual content desires some pre-processing earlier than we are able to teach the version. The best preprocessing we do here is - we delete commonly taking place words which we know aren't informative about the style. think of it as the litter in a few experience. those words are frequently removed and are called "stop phrases". can look them up on line. these encompass simple phrases like "a", "and", "but", "how", "or" and so on. They may be effortlessly eliminated using the python bundle NLTK.

From the above dataset, films with overviews which comprise handiest stop words, or movies with overviews containing no words with word2vec representation are unnoticed. Others are used to construct our mean word2vec representation. placed for every film overview

i)Take film overview

ii)Throw stop phrases

iii)For nonstop words:

tv)If in word2vec - take it is word2vec representation that is three hundred dimensional

If no longer - throw word

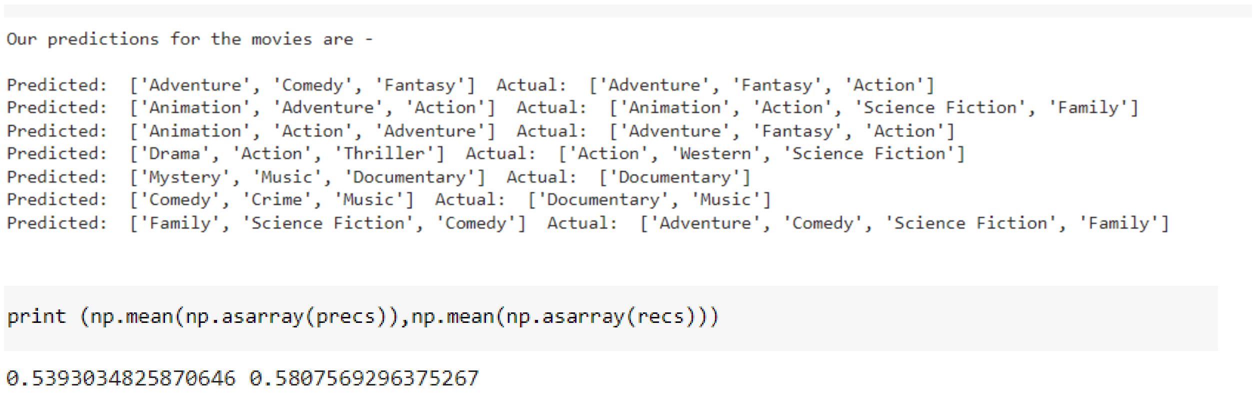
v)For each film, calculate the arithmetic imply of the 300-dimensional vector representations for all phrases in the assessment which weren't thrown out

300-dimensional representation for movies

. all these are stored in a NumPy array.

X matrix becomes (1553,300). And Y (1553,19) i.e., binarized 19 genres

[Appendix 1.9]



```
Our predictions for the movies are -

Predicted:  ['Adventure', 'Comedy', 'Fantasy']  Actual:  ['Adventure', 'Fantasy', 'Action']
Predicted:  ['Animation', 'Adventure', 'Action']  Actual:  ['Animation', 'Action', 'Science Fiction', 'Family']
Predicted:  ['Animation', 'Action', 'Adventure']  Actual:  ['Adventure', 'Fantasy', 'Action']
Predicted:  ['Drama', 'Action', 'Thriller']  Actual:  ['Action', 'Western', 'Science Fiction']
Predicted:  ['Mystery', 'Music', 'Documentary']  Actual:  ['Documentary']
Predicted:  ['Comedy', 'Crime', 'Music']  Actual:  ['Documentary', 'Music']
Predicted:  ['Family', 'Science Fiction', 'Comedy']  Actual:  ['Adventure', 'Comedy', 'Science Fiction', 'Family']
```

```
print (np.mean(np.asarray(precs)),np.mean(np.asarray(recs)))
```

```
0.5393034825870646 0.5807569296375267
```

these results can beat previous results.

got accuracies 78% when doing classification   scraped from Wikipedia.

The large amount of info very suitable for movie genre classification with a deep model.

<u>CONCLUSION</u>

In this project we have tried to solve the multimodal movie genre classification as a multi-label classification problem. To perform the classification, we used different sources of information, namely movie overview and poster of the movie. We scrapped data information from TMDb and IMDb APIs and build the data set from it. from the DENSNET 169 modal we achieved high F1 score. With the help of F1 score calculate we differentiated the movie genre class. We learned the technique clustering and bi-clustering with this regroup genre classes of the movie. We used the SVM and multinomial naïve bayes modal. We used deep learning to extract visual features from poster of the movies (VGCNET). Deep learning helps us understand the textual feature from overiew of the movie. The average precision is 0.53and recall score is 0.54 for our sample from SVM and Multinomial Naive Bayes modal. From word2vec average precision 0.539 and recall score 0.580.These models were trained on CPU's, and a simple 1-layer model was used to show that there is a lot of information in this data that the models can extract. With a larger dataset, and more training we able to bring these numbers to as high as 70%.

Reference

1. https://wikipedia.org/
2. Howard Zhou, Tucker Hermans, Asmita V Karandikar, and James M Rehg. Movie genre classification via scene categorization. In Proceedings of the 18th ACM international conference on Multimedia,2010.
3. Giuseppe Portolese and Valéria Delisandra Feltrim. On the use of synopsis-based features for film genre classification. In Anais do XV Encontro Nacional de Inteligência Artificial e Computacional,SBC, 2018.
4. Gabriel S Simões, Jônatas Wehrmann, Rodrigo C Barros, and Duncan D Ruiz. Movie genre classification with convolutional neural networks. In 2016 International Joint Conference on Neural Networks (IJCNN) IEEE, 2016.
5. Rafael C Gonzalez, Richard E Woods, et al. Digital image processing. Prentice hall Upper Saddle River, 2002.
6. Beth Logan et al. Mel frequency cepstral coefficients for music modeling. In ISMIR,2000.
7. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 1997.
8. Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. IEEE Transactions on Neural Networks and Learning Systems, 2017.

Github link:

https://github.com/TruthSearchers/ML_project

Appendix 1

1. data from IMDB amd TMDB through API

Here are steps to be done for data collection

- sign on for TMDB (themoviedatabase.org) and installation API to scrape film posters for above movies.
- set up and work with TMDb to get film information from their database
- Do the equal for IMDb
- examine the entries of IMDb and TMDb for a film
- Get a list and facts of a few films
- Get data from the TMDb

Sign up in TMDB and set up for getting movie metadata.

- Step1. Head over to [tmdb.org] (https://www.themoviedb.org/?language=en) and create a brand new account there by way of signing up.

- Step2. click on on account icon on the pinnacle proper, after that from drop down menu pick "Settings".

- Step3 at the settings page, will see the option "API" on the left pane. click on on that.

- Step4. follow for a brand new developer key. Fill out the shape as required. The fields "application call" and "application URL" are not essential. Fill whatever there.

- Step five. It must generate a new API key for and should additionally receive a mail.

Now which have the API key for TMDB, can question the use of TMDB. remember, it permits best 40 queries per 10 seconds.

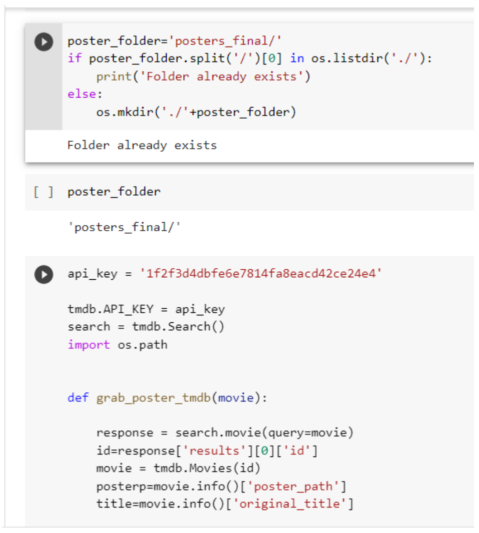
An easy way to admire this is to simply have a name to time. Sleep (1) after each new release. that is additionally being very ok to the server.

If want to try and maximize throughput can embed each TMDB request in a nested, strive besides block. If the first strive fails, the second try first uses python's sleep characteristic to offer it a little relaxation, and after that attempt once more to make a request. some thing like this -

Using TMDB to get movie information with API

I TMDBsimple which makes TMDB using even easier. This library was installed at the time of setup.

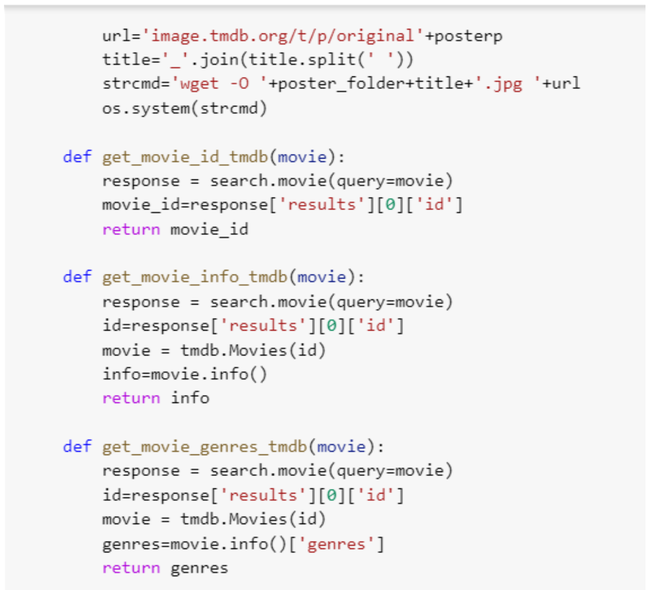
```
url = 'https://api.themoviedb.org/3/movie/1581?api_key=' + api_key
data = urllib2.urlopen(url). read ()
```



```
poster_folder='posters_final/'
if poster_folder.split('/')[0] in os.listdir('./'):
    print('Folder already exists')
else:
    os.mkdir('./'+poster_folder)

Folder already exists
```

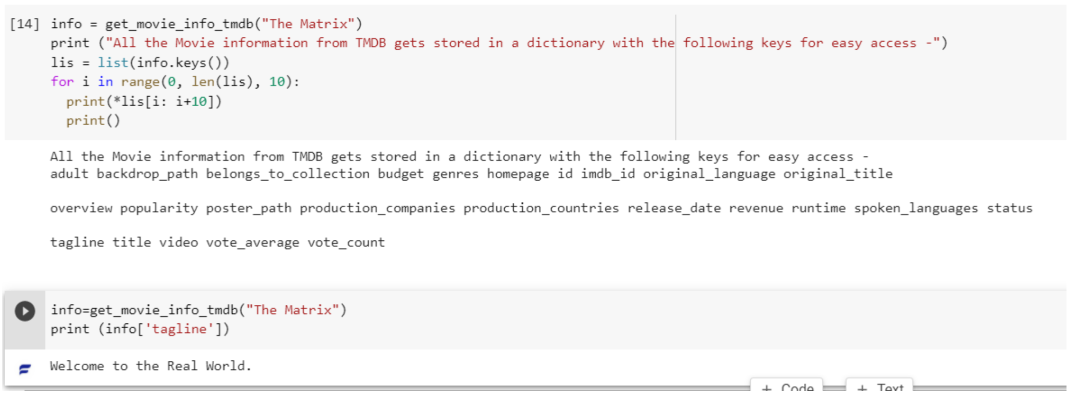
```
[ ] poster_folder

'posters_final/'
```

```
api_key = '1f2f3d4dbfe6e7814fa8eacd42ce24e4'

tmdb.API_KEY = api_key
search = tmdb.Search()
import os.path


def grab_poster_tmdb(movie):

    response = search.movie(query=movie)
    id=response['results'][0]['id']
    movie = tmdb.Movies(id)
    posterp=movie.info()['poster_path']
    title=movie.info()['original_title']
```



```
    url='image.tmdb.org/t/p/original'+posterp
    title='_'.join(title.split(' '))
    strcmd='wget -O '+poster_folder+title+'.jpg '+url
    os.system(strcmd)

def get_movie_id_tmdb(movie):
    response = search.movie(query=movie)
    movie_id=response['results'][0]['id']
    return movie_id

def get_movie_info_tmdb(movie):
    response = search.movie(query=movie)
    id=response['results'][0]['id']
    movie = tmdb.Movies(id)
    info=movie.info()
    return info

def get_movie_genres_tmdb(movie):
    response = search.movie(query=movie)
    id=response['results'][0]['id']
    movie = tmdb.Movies(id)
    genres=movie.info()['genres']
    return genres
```

while the above features were made to make it easy to get genres, posters, and id, all of the information that can be accessed may be visible via calling the function get_movie_info () as shown underneath

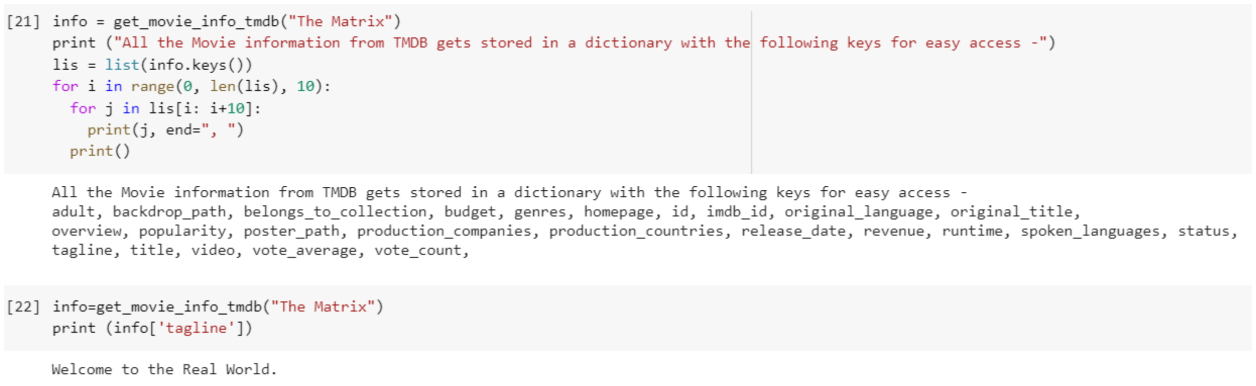


```
[14] info = get_movie_info_tmdb("The Matrix")
     print ("All the Movie information from TMDB gets stored in a dictionary with the following keys for easy access -")
     lis = list(info.keys())
     for i in range(0, len(lis), 10):
       print(*lis[i: i+10])
       print()

     All the Movie information from TMDB gets stored in a dictionary with the following keys for easy access -
     adult backdrop_path 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
```

```
     info=get_movie_info_tmdb("The Matrix")
     print (info['tagline'])

     Welcome to the Real World.
```

Get movie information through IMDB

Now that we recognise the way to get information from TMDB, here's how we can get statistics approximately the identical movie from IMDB. This makes it viable for us to mix more data and get a richer dataset.. due to the variations between the 2 datasets, will must perform a little cleaning, however each datasets are extraordinarily clean, and it will likely be minimum.

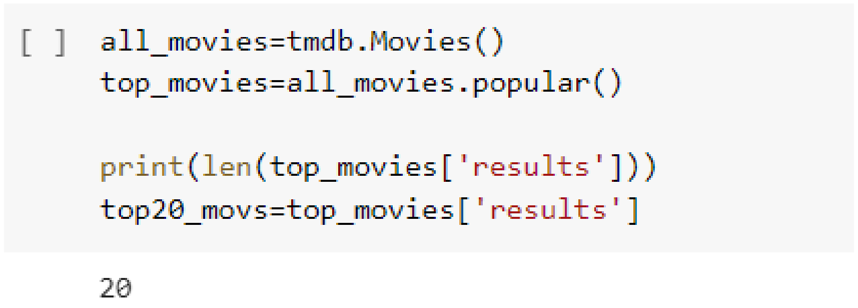


```
[21] info = get_movie_info_tmdb("The Matrix")
     print ("All the Movie information from TMDB gets stored in a dictionary with the following keys for easy access -")
     lis = list(info.keys())
     for i in range(0, len(lis), 10):
       for j in lis[i: i+10]:
         print(j, end=", ")
       print()

     All the Movie information from TMDB gets stored in a dictionary with the following keys for easy access -
     adult, backdrop_path, 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,
```

```
[22] info=get_movie_info_tmdb("The Matrix")
     print (info['tagline'])

     Welcome to the Real World.
```

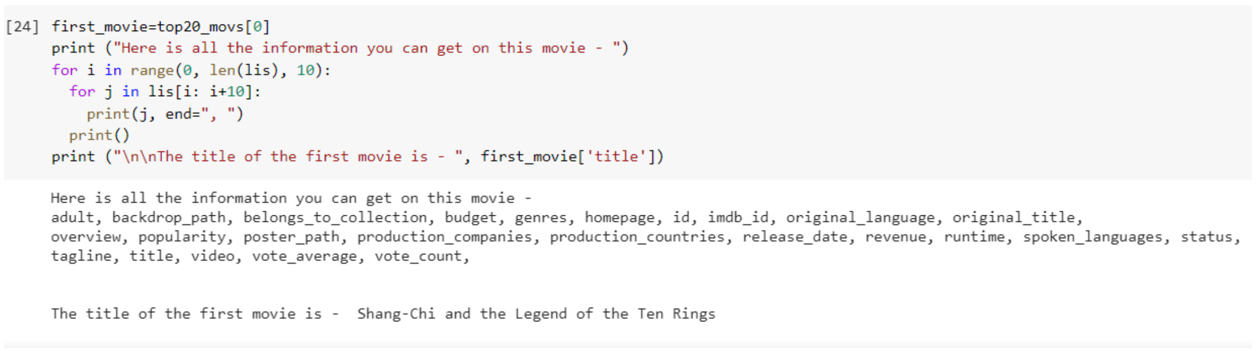
2. Call data from API for checking of the data and properties in our data

Obtaining Top 20 movies from TMDB



```
[ ]  all_movies=tmdb.Movies()
     top_movies=all_movies.popular()

     print(len(top_movies['results']))
     top20_movs=top_movies['results']

     20
```

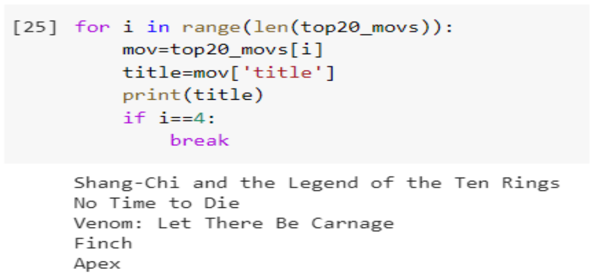
as we had information on the movie "The Matrix", as can see below. A dictionary which can be queried for specific information on movie



```
[24] first_movie=top20_movs[0]
     print ("Here is all the information you can get on this movie - ")
     for i in range(0, len(lis), 10):
       for j in lis[i: i+10]:
         print(j, end=", ")
       print()
     print ("\n\nThe title of the first movie is - ", first_movie['title'])
```

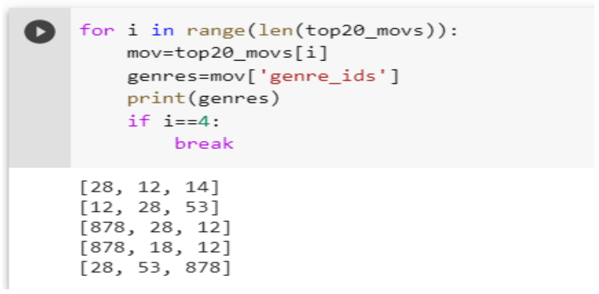
```
Here is all the information you can get on this movie -
adult, backdrop_path, belongs_to_collection, budget, genres, homepage, id, imdb_id, original_language, original_title,
overview, popularity, poster_path, production_companies, production_countries, release_date, revenue, runtime, spoken_languages, status,
tagline, title, video, vote_average, vote_count,


The title of the first movie is -  Shang-Chi and the Legend of the Ten Rings
```

Moving on, we can get genres in the same way.



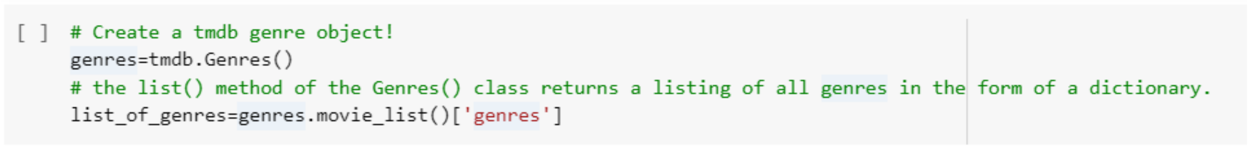
```
[25] for i in range(len(top20_movs)):
         mov=top20_movs[i]
         title=mov['title']
         print(title)
         if i==4:
             break
```

```
Shang-Chi and the Legend of the Ten Rings
No Time to Die
Venom: Let There Be Carnage
Finch
Apex
```



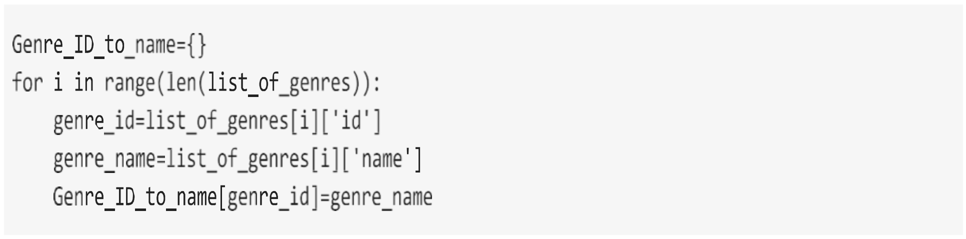
```
for i in range(len(top20_movs)):
    mov=top20_movs[i]
    genres=mov['genre_ids']
    print(genres)
    if i==4:
        break
```

```
[28, 12, 14]
[12, 28, 53]
[878, 28, 12]
[878, 18, 12]
[28, 53, 878]
```

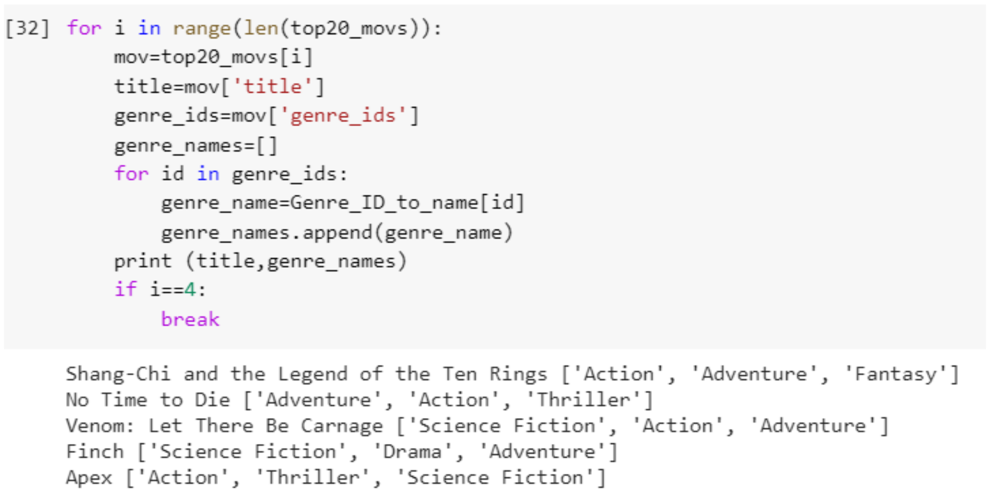


```
[ ]  # Create a tmdb genre object!
     genres=tmdb.Genres()
     # the list() method of the Genres() class returns a listing of all genres in the form of a dictionary.
     list_of_genres=genres.movie_list()['genres']
```

converting list into a nice dictionary to look up genre names from itsIDs!



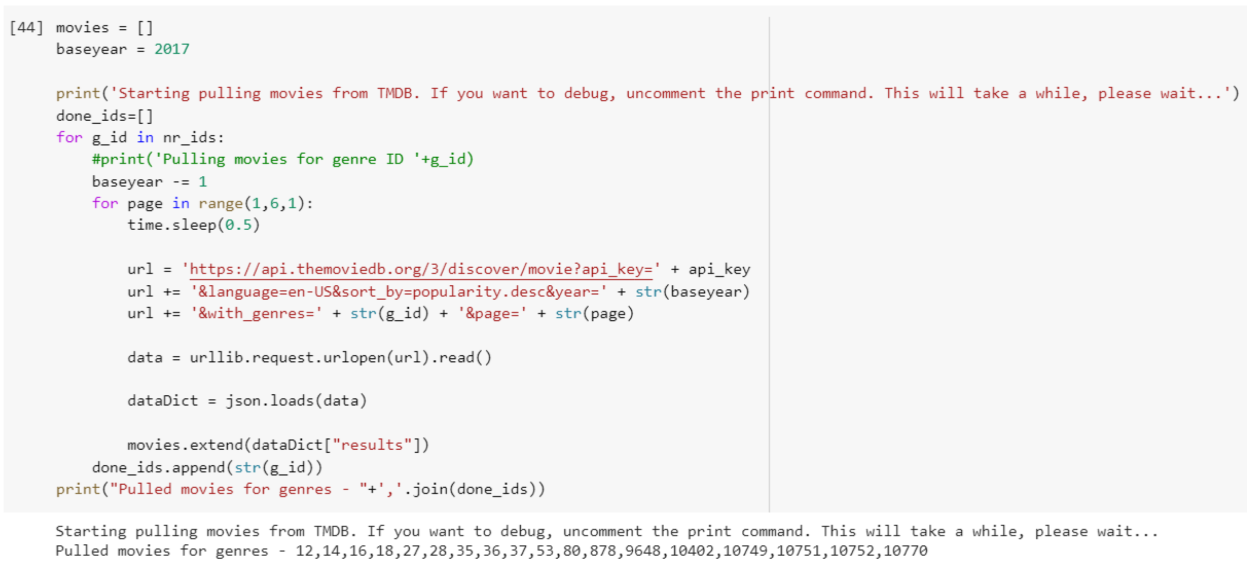
```
Genre_ID_to_name={}
for i in range(len(list_of_genres)):
    genre_id=list_of_genres[i]['id']
    genre_name=list_of_genres[i]['name']
    Genre_ID_to_name[genre_id]=genre_name
```



```
[32] for i in range(len(top20_movs)):
         mov=top20_movs[i]
         title=mov['title']
         genre_ids=mov['genre_ids']
         genre_names=[]
         for id in genre_ids:
             genre_name=Genre_ID_to_name[id]
             genre_names.append(genre_name)
         print (title,genre_names)
         if i==4:
             break
```

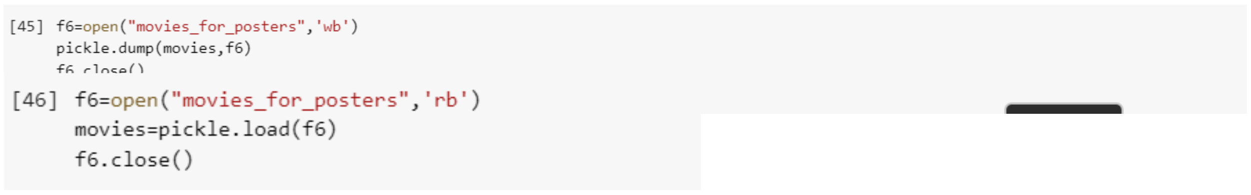
```
Shang-Chi and the Legend of the Ten Rings ['Action', 'Adventure', 'Fantasy']
No Time to Die ['Adventure', 'Action', 'Thriller']
Venom: Let There Be Carnage ['Science Fiction', 'Action', 'Adventure']
Finch ['Science Fiction', 'Drama', 'Adventure']
Apex ['Action', 'Thriller', 'Science Fiction']
```

3. Pulling posters from TMDB for our training data



```
[44] movies = []
     baseyear = 2017

     print('Starting pulling movies from TMDB. If you want to debug, uncomment the print command. This will take a while, please wait...')
     done_ids=[]
     for g_id in nr_ids:
         #print('Pulling movies for genre ID '+g_id)
         baseyear -= 1
         for page in range(1,6,1):
             time.sleep(0.5)

             url = 'https://api.themoviedb.org/3/discover/movie?api_key=' + api_key
             url += '&language=en-US&sort_by=popularity.desc&year=' + str(baseyear)
             url += '&with_genres=' + str(g_id) + '&page=' + str(page)

             data = urllib.request.urlopen(url).read()

             dataDict = json.loads(data)

             movies.extend(dataDict["results"])
         done_ids.append(str(g_id))
     print("Pulled movies for genres - "+','.join(done_ids))
```

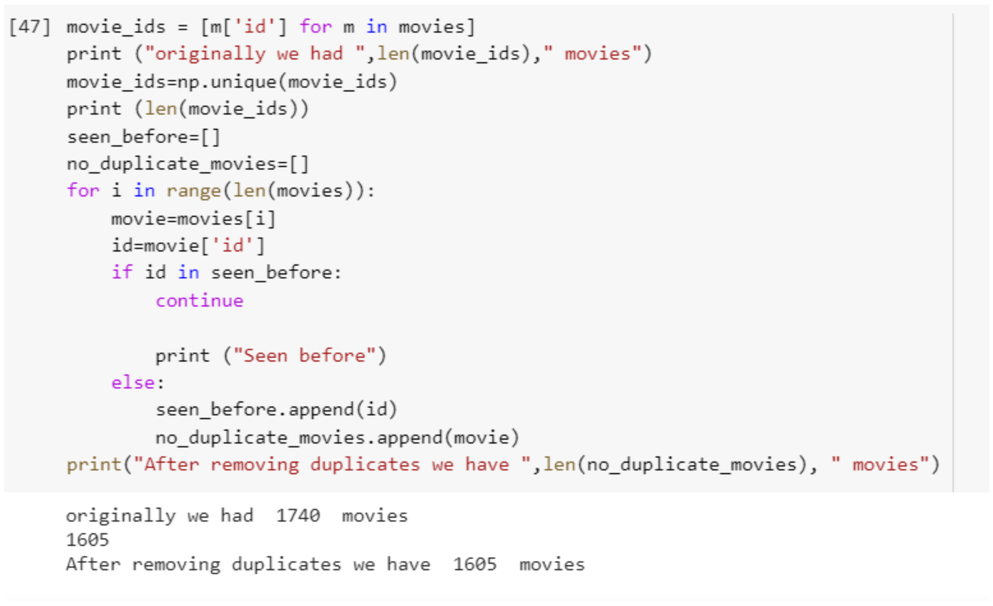
```
     Starting pulling movies from TMDB. If you want to debug, uncomment the print command. This will take a while, please wait...
     Pulled movies for genres - 12,14,16,18,27,28,35,36,37,53,80,878,9648,10402,10749,10751,10752,10770
```



```
[45] f6=open("movies_for_posters",'wb')
     pickle.dump(movies,f6)
     f6.close()
```

```
[46] f6=open("movies_for_posters",'rb')
     movies=pickle.load(f6)
     f6.close()
```

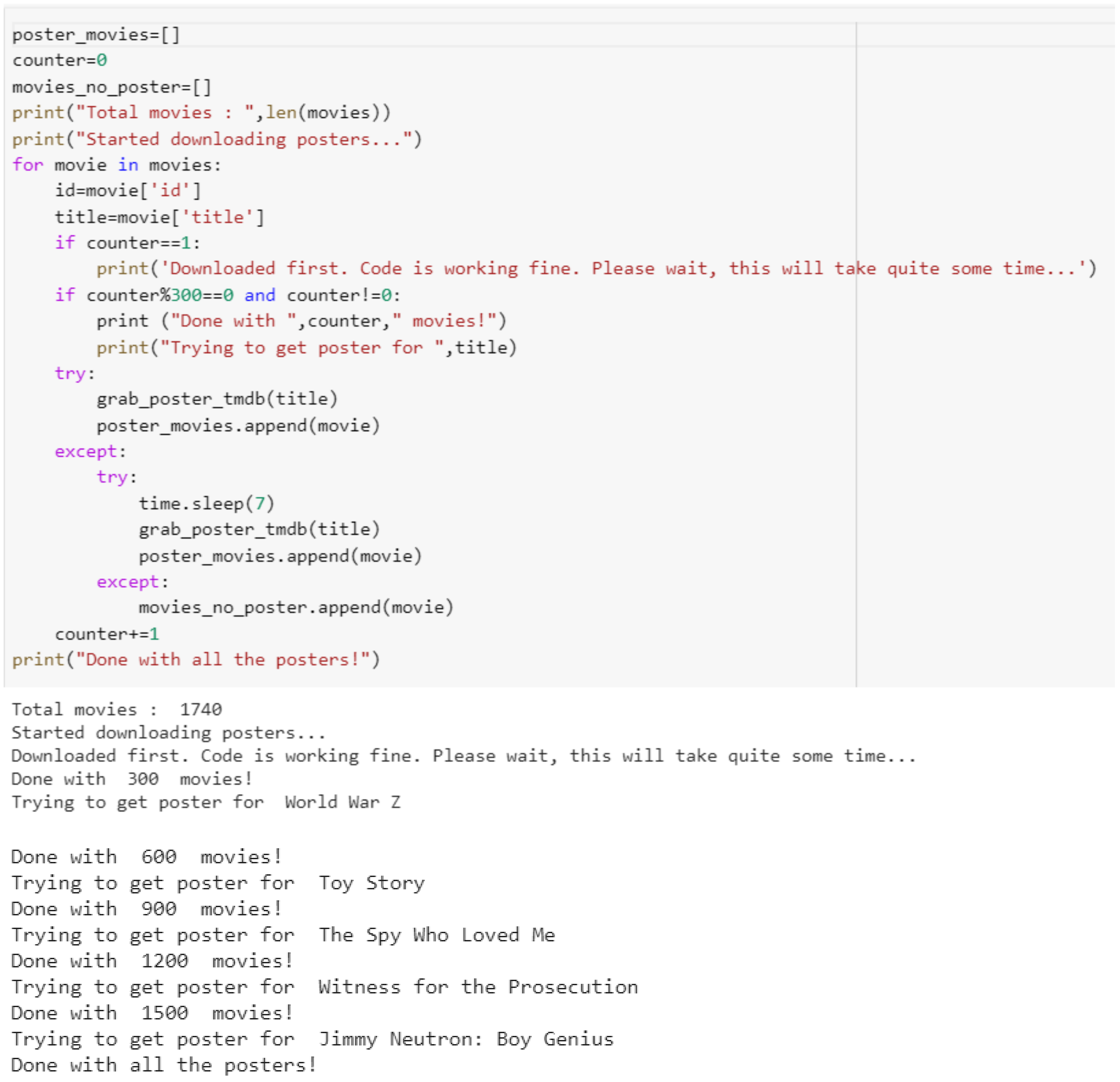
Let's remove duplicates from the list of movies



```
[47] movie_ids = [m['id'] for m in movies]
     print ("originally we had ",len(movie_ids)," movies")
     movie_ids=np.unique(movie_ids)
     print (len(movie_ids))
     seen_before=[]
     no_duplicate_movies=[]
     for i in range(len(movies)):
         movie=movies[i]
         id=movie['id']
         if id in seen_before:
             continue

             print ("Seen before")
         else:
             seen_before.append(id)
             no_duplicate_movies.append(movie)
     print("After removing duplicates we have ",len(no_duplicate_movies), " movies")
```

```
     originally we had  1740  movies
     1605
     After removing duplicates we have  1605  movies
```

Also, let's remove movies for which we have no posters!



```python
poster_movies=[]
counter=0
movies_no_poster=[]
print("Total movies : ",len(movies))
print("Started downloading posters...")
for movie in movies:
    id=movie['id']
    title=movie['title']
    if counter==1:
        print('Downloaded first. Code is working fine. Please wait, this will take quite some time...')
    if counter%300==0 and counter!=0:
        print ("Done with ",counter," movies!")
        print("Trying to get poster for ",title)
    try:
        grab_poster_tmdb(title)
        poster_movies.append(movie)
    except:
        try:
            time.sleep(7)
            grab_poster_tmdb(title)
            poster_movies.append(movie)
        except:
            movies_no_poster.append(movie)
    counter+=1
print("Done with all the posters!")
```

```
Total movies :  1740
Started downloading posters...
Downloaded first. Code is working fine. Please wait, this will take quite some time...
Done with  300  movies!
Trying to get poster for  World War Z

Done with  600  movies!
Trying to get poster for  Toy Story
Done with  900  movies!
Trying to get poster for  The Spy Who Loved Me
Done with  1200  movies!
Trying to get poster for  Witness for the Prosecution
Done with  1500  movies!
Trying to get poster for  Jimmy Neutron: Boy Genius
Done with all the posters!
```

```
[56] print (len(movies_no_poster))
     print (len(poster_movies))

     27
     1713
```

```
[58] f=open('poster_movies.pckl','wb')
     pickle.dump(poster_movies,f)
     f.close()
```

```
[60] f=open('poster_movies.pckl','rb')
     poster_movies=pickle.load(f)
     f.close()
```

```
[62] f=open('no_poster_movies.pckl','wb')
     pickle.dump(movies_no_poster,f)
     f.close()
```

```
[64] f=open('no_poster_movies.pckl','rb')
     movies_no_poster=pickle.load(f)
     f.close()
```

4. Building X and Y

```
[68] genres=[]
     all_ids=[]
     for i in range(len(movies_with_overviews)):
         movie=movies_with_overviews[i]
         id=movie['id']
         genre_ids=movie['genre_ids']
         genres.append(genre_ids)
         all_ids.extend(genre_ids)
```

```
[70] from sklearn.preprocessing import MultiLabelBinarizer
     mlb=MultiLabelBinarizer()
     Y=mlb.fit_transform(genres)
```

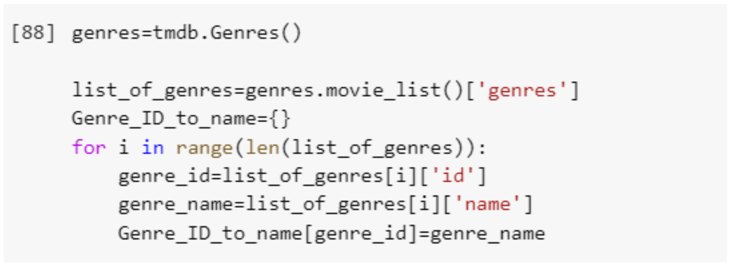
```
[69] genres[1]

     [14, 12]
```

```
[71] print(Y.shape)
     print (np.sum(Y, axis=0))

     (1600, 19)
     [388 286 301 642 202 494 432 151  72 412 212  35 256 184 127 240 303 131
      128]
```

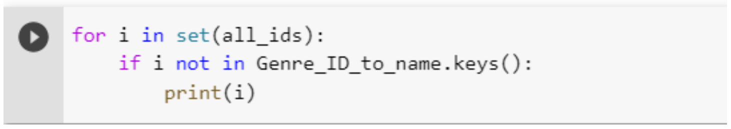
```
len(list_of_genres)

19
```

Staryed with only 19 genre labels if remember. However the shape for Y is 1693,19 while it should be 1666,19 as there are only 19 genres

finding genre IDs not present in original list of genres!

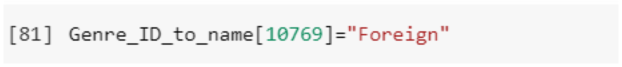


```
[88] genres=tmdb.Genres()

     list_of_genres=genres.movie_list()['genres']
     Genre_ID_to_name={}
     for i in range(len(list_of_genres)):
         genre_id=list_of_genres[i]['id']
         genre_name=list_of_genres[i]['name']
         Genre_ID_to_name[genre_id]=genre_name
```

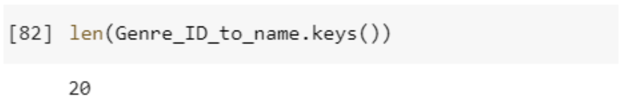


```
for i in set(all_ids):
    if i not in Genre_ID_to_name.keys():
        print(i)
```

This genre id wasn't given to us by means of TMDB when we asked it for all possible genres. we can either forget all samples that have this style. however if appearance up 'll see there may be too lots of those samples. So, we googled extra and went into their documentation and observed that this identification corresponds to the genre "foreign" So, we add it to the dictionary of genre names ourselves. Such issues are ubiquitous in gadget gaining knowledge of, and it's far up to us to diagnose and accurate them. We have to continually determine approximately what to hold, the way to save data and so forth.

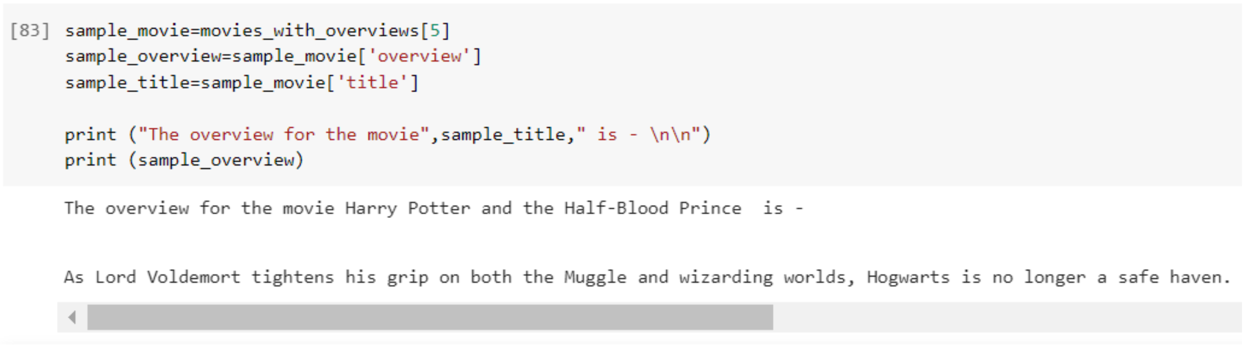


```
[81] Genre_ID_to_name[10769]="Foreign"
```



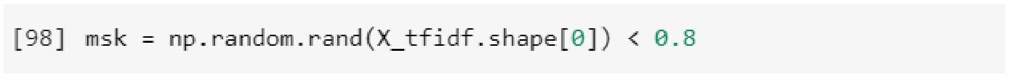
```
[82] len(Genre_ID_to_name.keys())

     20
```

Now, turn to building the the input features! we will be using the overview of movies as our input vector! eg

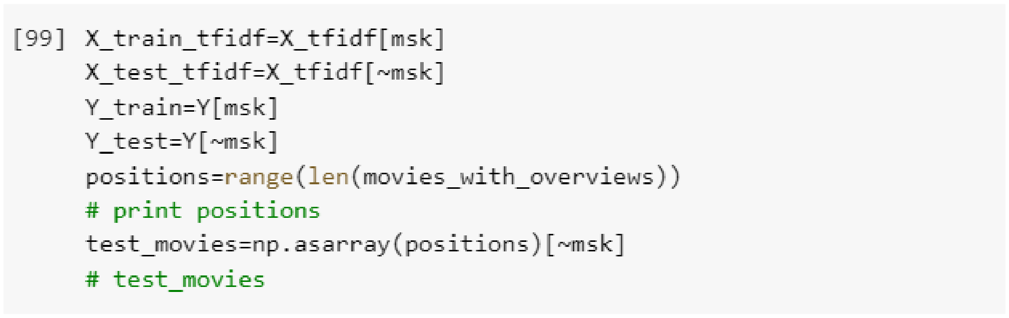


```
[83] sample_movie=movies_with_overviews[5]
     sample_overview=sample_movie['overview']
     sample_title=sample_movie['title']

     print ("The overview for the movie",sample_title," is - \n\n")
     print (sample_overview)

     The overview for the movie Harry Potter and the Half-Blood Prince  is -


     As Lord Voldemort tightens his grip on both the Muggle and wizarding worlds, Hogwarts is no longer a safe haven.
```

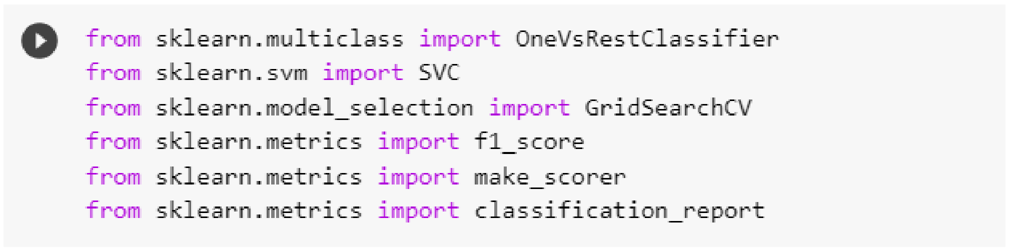
5. Training X & Y



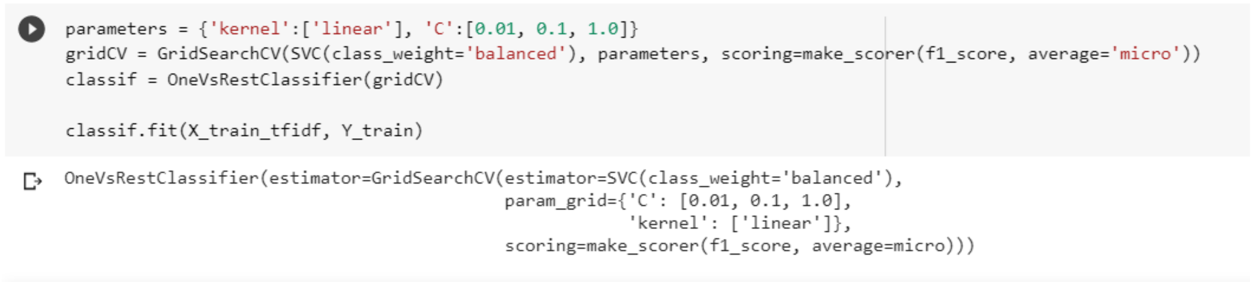
```
[98] msk = np.random.rand(X_tfidf.shape[0]) < 0.8
```



```
[99] X_train_tfidf=X_tfidf[msk]
     X_test_tfidf=X_tfidf[~msk]
     Y_train=Y[msk]
     Y_test=Y[~msk]
     positions=range(len(movies_with_overviews))
     # print positions
     test_movies=np.asarray(positions)[~msk]
     # test_movies
```

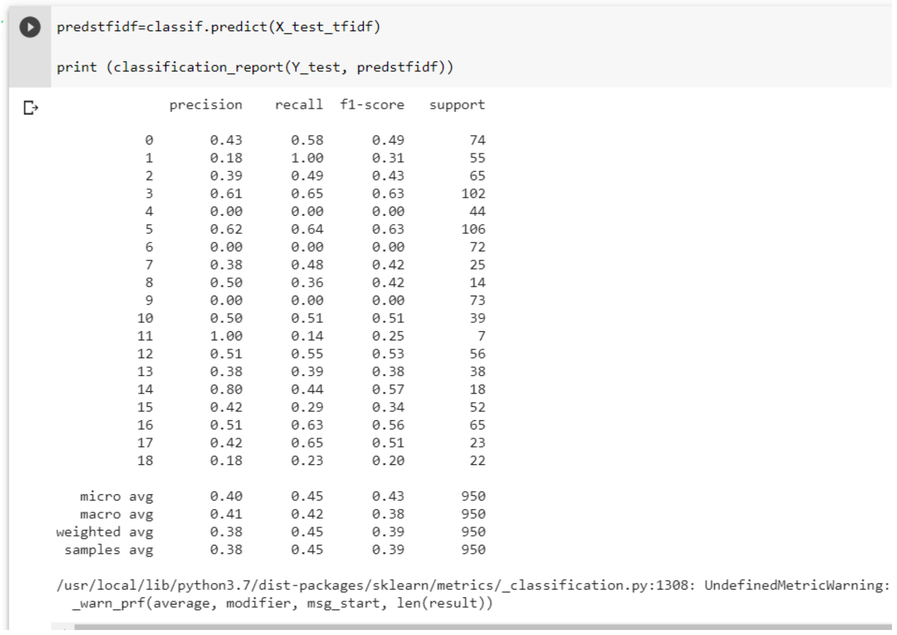


```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer
from sklearn.metrics import classification_report
```



```
parameters = {'kernel':['linear'], 'C':[0.01, 0.1, 1.0]}
gridCV = GridSearchCV(SVC(class_weight='balanced'), parameters, scoring=make_scorer(f1_score, average='micro'))
classif = OneVsRestClassifier(gridCV)

classif.fit(X_train_tfidf, Y_train)
```

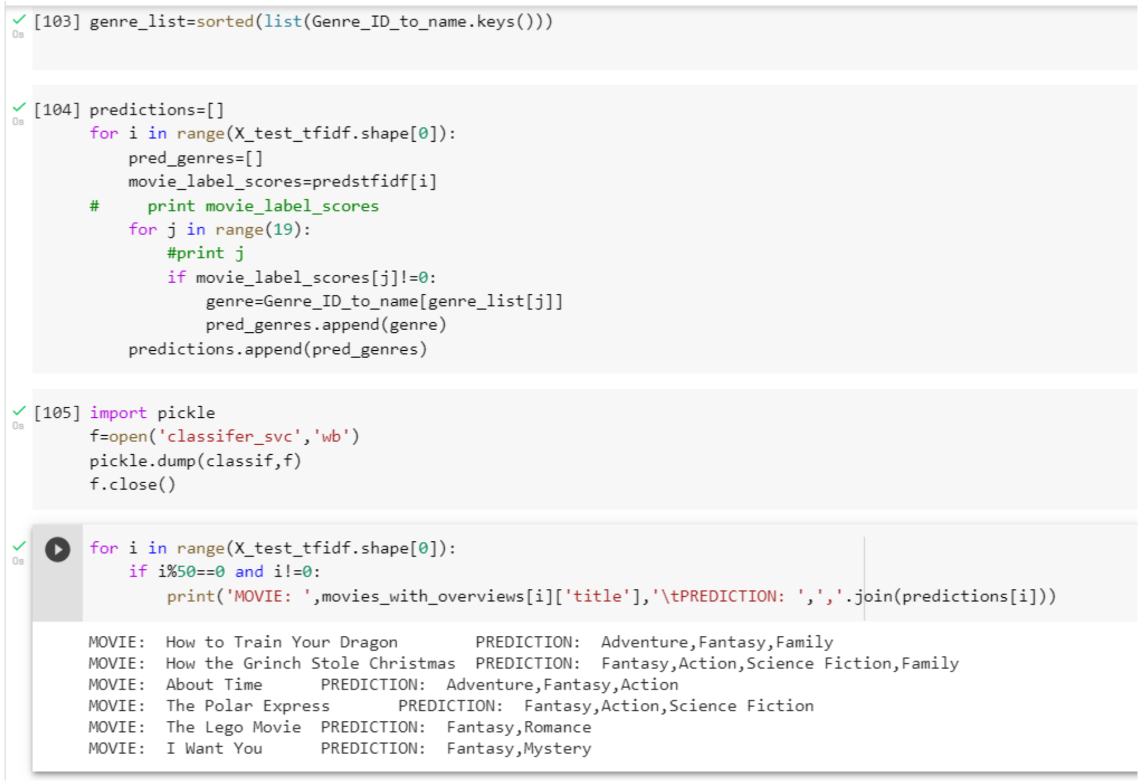
```
OneVsRestClassifier(estimator=GridSearchCV(estimator=SVC(class_weight='balanced'),
                                           param_grid={'C': [0.01, 0.1, 1.0],
                                                       'kernel': ['linear']},
                                           scoring=make_scorer(f1_score, average=micro)))
```



```
predstfidf=classif.predict(X_test_tfidf)

print (classification_report(Y_test, predstfidf))
```

```
              precision    recall  f1-score   support

           0       0.43      0.58      0.49        74
           1       0.18      1.00      0.31        55
           2       0.39      0.49      0.43        65
           3       0.61      0.65      0.63       102
           4       0.00      0.00      0.00        44
           5       0.62      0.64      0.63       106
           6       0.00      0.00      0.00        72
           7       0.38      0.48      0.42        25
           8       0.50      0.36      0.42        14
           9       0.00      0.00      0.00        73
          10       0.50      0.51      0.51        39
          11       1.00      0.14      0.25         7
          12       0.51      0.55      0.53        56
          13       0.38      0.39      0.38        38
          14       0.80      0.44      0.57        18
          15       0.42      0.29      0.34        52
          16       0.51      0.63      0.56        65
          17       0.42      0.65      0.51        23
          18       0.18      0.23      0.20        22

   micro avg       0.40      0.45      0.43       950
   macro avg       0.41      0.42      0.38       950
weighted avg       0.38      0.45      0.39       950
 samples avg       0.38      0.45      0.39       950

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning:
  _warn_prf(average, modifier, msg_start, len(result))
```

6. Evaluating metric



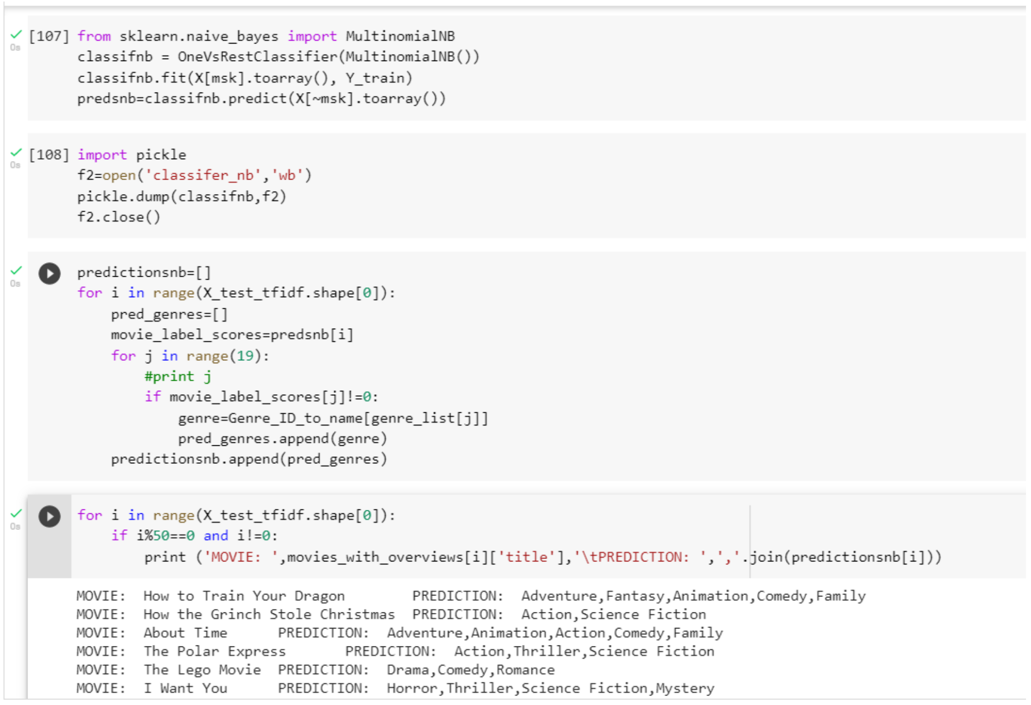
```
[103] genre_list=sorted(list(Genre_ID_to_name.keys()))
```

```
[104] predictions=[]
      for i in range(X_test_tfidf.shape[0]):
          pred_genres=[]
          movie_label_scores=predstfidf[i]
      #     print movie_label_scores
          for j in range(19):
              #print j
              if movie_label_scores[j]!=0:
                  genre=Genre_ID_to_name[genre_list[j]]
                  pred_genres.append(genre)
          predictions.append(pred_genres)
```

```
[105] import pickle
      f=open('classifer_svc','wb')
      pickle.dump(classif,f)
      f.close()
```

```
for i in range(X_test_tfidf.shape[0]):
    if i%50==0 and i!=0:
        print('MOVIE: ',movies_with_overviews[i]['title'],'\tPREDICTION: ',','.join(predictions[i]))
```

```
MOVIE:  How to Train Your Dragon        PREDICTION:  Adventure,Fantasy,Family
MOVIE:  How the Grinch Stole Christmas  PREDICTION:  Fantasy,Action,Science Fiction,Family
MOVIE:  About Time      PREDICTION:  Adventure,Fantasy,Action
MOVIE:  The Polar Express       PREDICTION:  Fantasy,Action,Science Fiction
MOVIE:  The Lego Movie  PREDICTION:  Fantasy,Romance
MOVIE:  I Want You      PREDICTION:  Fantasy,Mystery
```



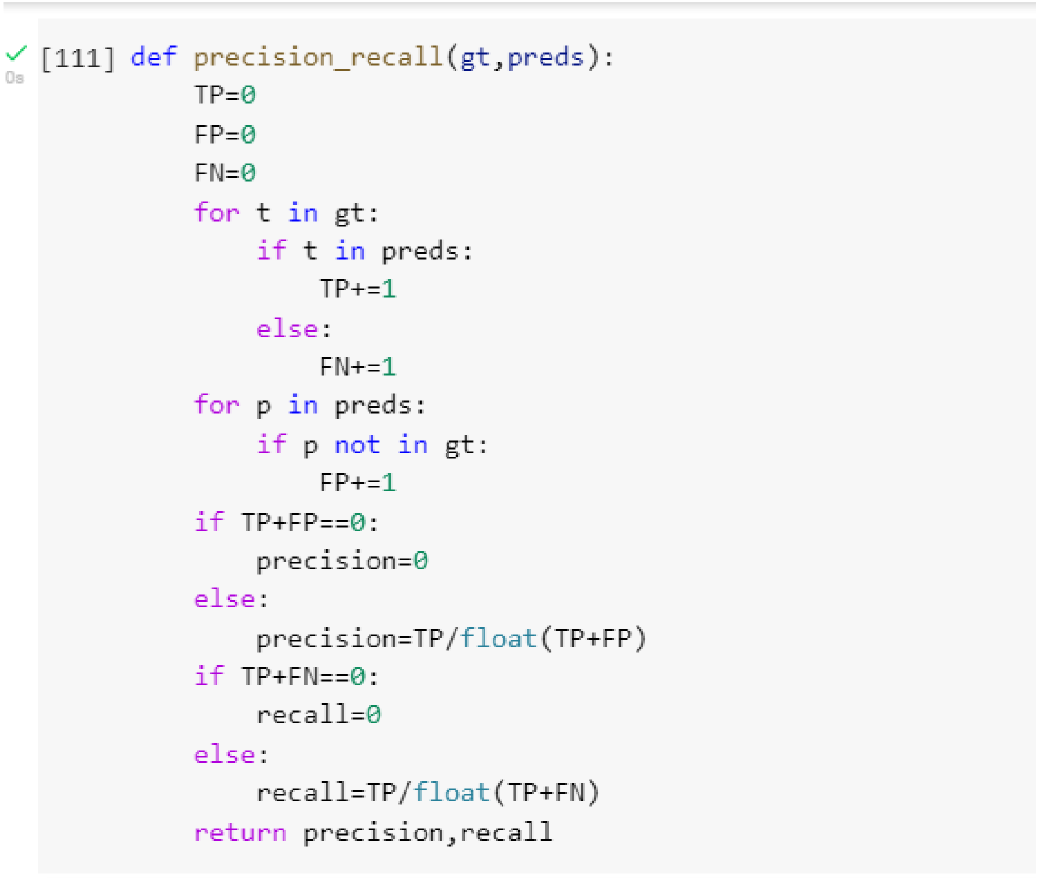
```
[107] from sklearn.naive_bayes import MultinomialNB
      classifnb = OneVsRestClassifier(MultinomialNB())
      classifnb.fit(X[msk].toarray(), Y_train)
      predsnb=classifnb.predict(X[~msk].toarray())
```

```
[108] import pickle
      f2=open('classifer_nb','wb')
      pickle.dump(classifnb,f2)
      f2.close()
```

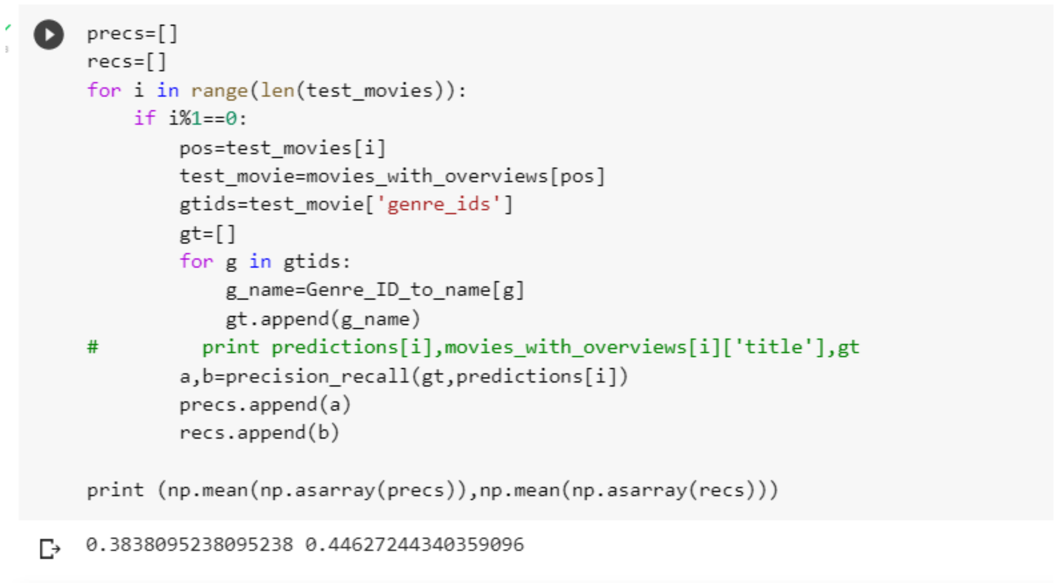
```
predictionsnb=[]
for i in range(X_test_tfidf.shape[0]):
    pred_genres=[]
    movie_label_scores=predsnb[i]
    for j in range(19):
        #print j
        if movie_label_scores[j]!=0:
            genre=Genre_ID_to_name[genre_list[j]]
            pred_genres.append(genre)
    predictionsnb.append(pred_genres)
```

```
for i in range(X_test_tfidf.shape[0]):
    if i%50==0 and i!=0:
        print ('MOVIE: ',movies_with_overviews[i]['title'],'\tPREDICTION: ',','.join(predictionsnb[i]))
```

```
MOVIE:  How to Train Your Dragon        PREDICTION:  Adventure,Fantasy,Animation,Comedy,Family
MOVIE:  How the Grinch Stole Christmas  PREDICTION:  Action,Science Fiction
MOVIE:  About Time      PREDICTION:  Adventure,Animation,Action,Comedy,Family
MOVIE:  The Polar Express       PREDICTION:  Action,Thriller,Science Fiction
MOVIE:  The Lego Movie  PREDICTION:  Drama,Comedy,Romance
MOVIE:  I Want You      PREDICTION:  Horror,Thriller,Science Fiction,Mystery
```

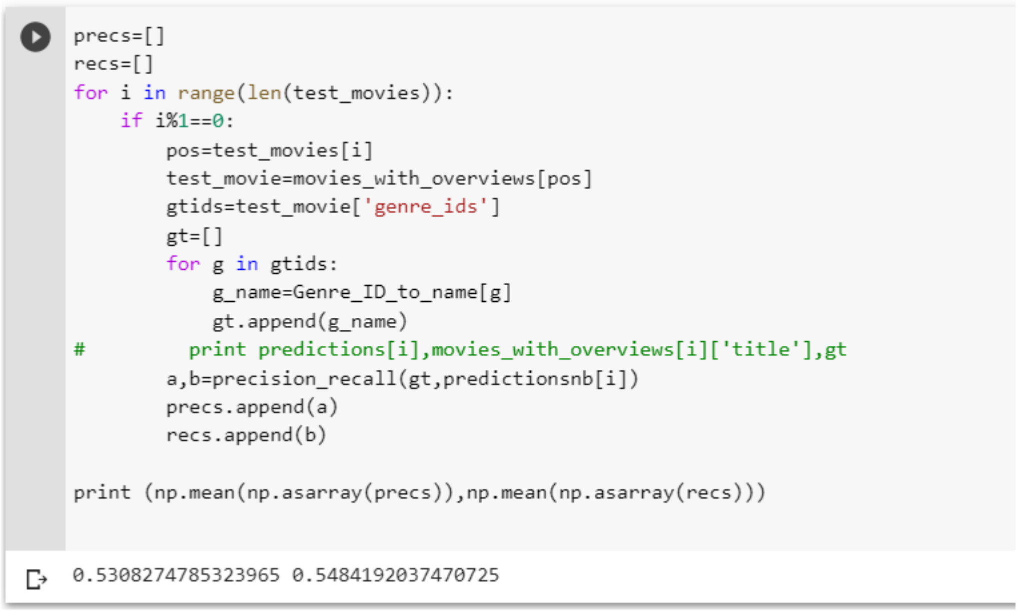


```
[111] def precision_recall(gt,preds):
          TP=0
          FP=0
          FN=0
          for t in gt:
              if t in preds:
                  TP+=1
              else:
                  FN+=1
          for p in preds:
              if p not in gt:
                  FP+=1
          if TP+FP==0:
              precision=0
          else:
              precision=TP/float(TP+FP)
          if TP+FN==0:
              recall=0
          else:
              recall=TP/float(TP+FN)
          return precision,recall
```



```
precs=[]
recs=[]
for i in range(len(test_movies)):
    if i%1==0:
        pos=test_movies[i]
        test_movie=movies_with_overviews[pos]
        gtids=test_movie['genre_ids']
        gt=[]
        for g in gtids:
            g_name=Genre_ID_to_name[g]
            gt.append(g_name)
#         print predictions[i],movies_with_overviews[i]['title'],gt
        a,b=precision_recall(gt,predictions[i])
        precs.append(a)
        recs.append(b)

print (np.mean(np.asarray(precs)),np.mean(np.asarray(recs)))
```

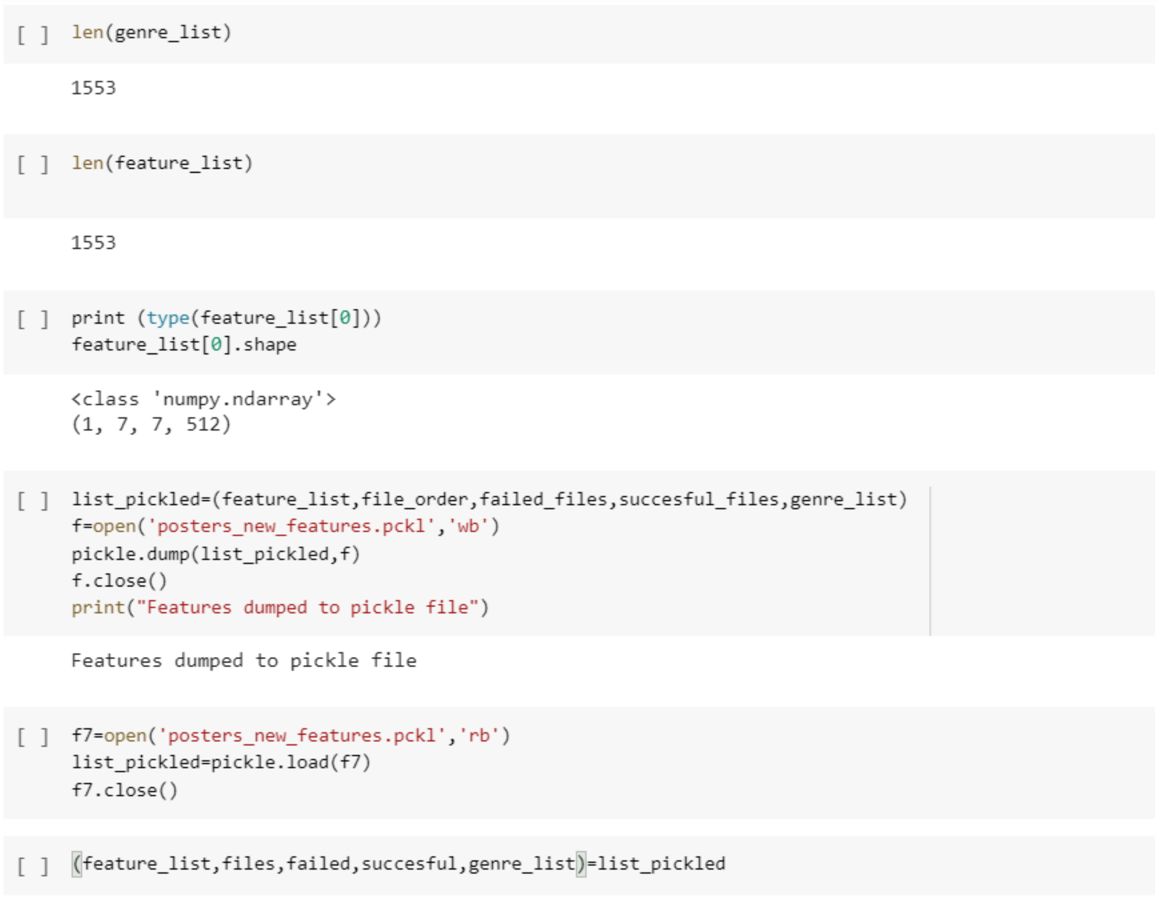
```
0.3838095238095238 0.44627244340359096
```



```
precs=[]
recs=[]
for i in range(len(test_movies)):
    if i%1==0:
        pos=test_movies[i]
        test_movie=movies_with_overviews[pos]
        gtids=test_movie['genre_ids']
        gt=[]
        for g in gtids:
            g_name=Genre_ID_to_name[g]
            gt.append(g_name)
#         print predictions[i],movies_with_overviews[i]['title'],gt
        a,b=precision_recall(gt,predictionsnb[i])
        precs.append(a)
        recs.append(b)

print (np.mean(np.asarray(precs)),np.mean(np.asarray(recs)))
```

```
0.5308274785323965 0.5484192037470725
```

7. Training neural network model using VGG features.



```
[ ] len(genre_list)
```

```
1553
```

```
[ ] len(feature_list)
```

```
1553
```

```
[ ] print (type(feature_list[0]))
    feature_list[0].shape
```

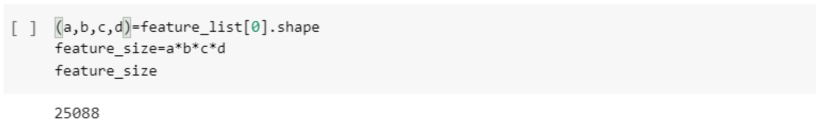
```
<class 'numpy.ndarray'>
(1, 7, 7, 512)
```

```
[ ] list_pickled=(feature_list,file_order,failed_files,succesful_files,genre_list)
    f=open('posters_new_features.pckl','wb')
    pickle.dump(list_pickled,f)
    f.close()
    print("Features dumped to pickle file")
```

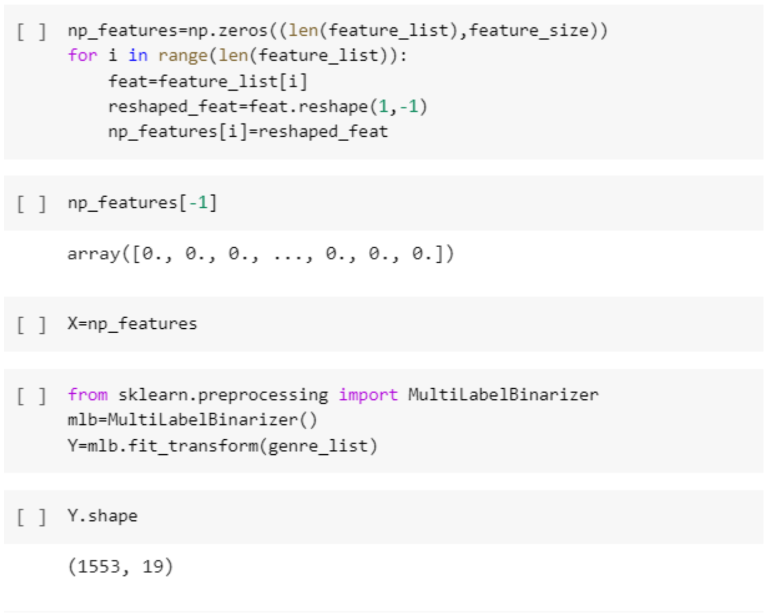
```
Features dumped to pickle file
```

```
[ ] f7=open('posters_new_features.pckl','rb')
    list_pickled=pickle.load(f7)
    f7.close()
```

```
[ ] (feature_list,files,failed,succesful,genre_list)=list_pickled
```

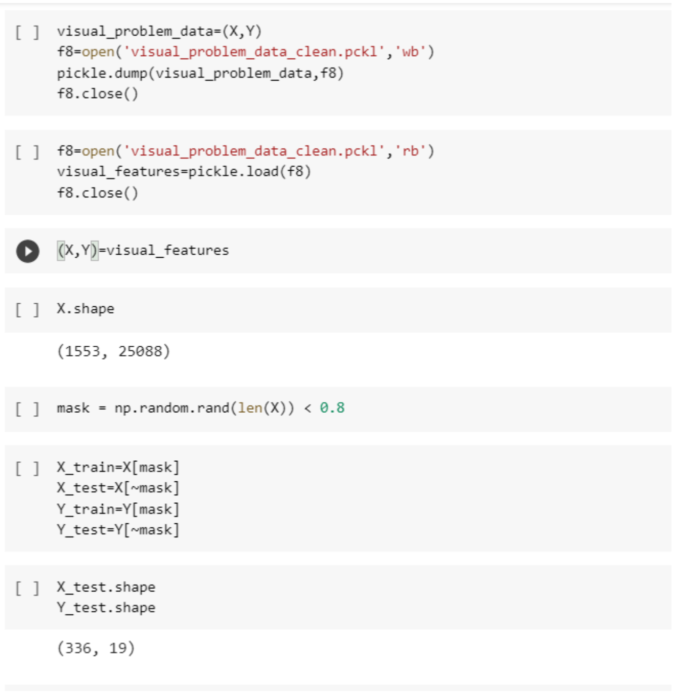


```
[ ]  (a,b,c,d)=feature_list[0].shape
     feature_size=a*b*c*d
     feature_size

     25088
```



```
[ ]  np_features=np.zeros((len(feature_list),feature_size))
     for i in range(len(feature_list)):
         feat=feature_list[i]
         reshaped_feat=feat.reshape(1,-1)
         np_features[i]=reshaped_feat

[ ]  np_features[-1]

     array([0., 0., 0., ..., 0., 0., 0.])

[ ]  X=np_features

[ ]  from sklearn.preprocessing import MultiLabelBinarizer
     mlb=MultiLabelBinarizer()
     Y=mlb.fit_transform(genre_list)

[ ]  Y.shape

     (1553, 19)
```

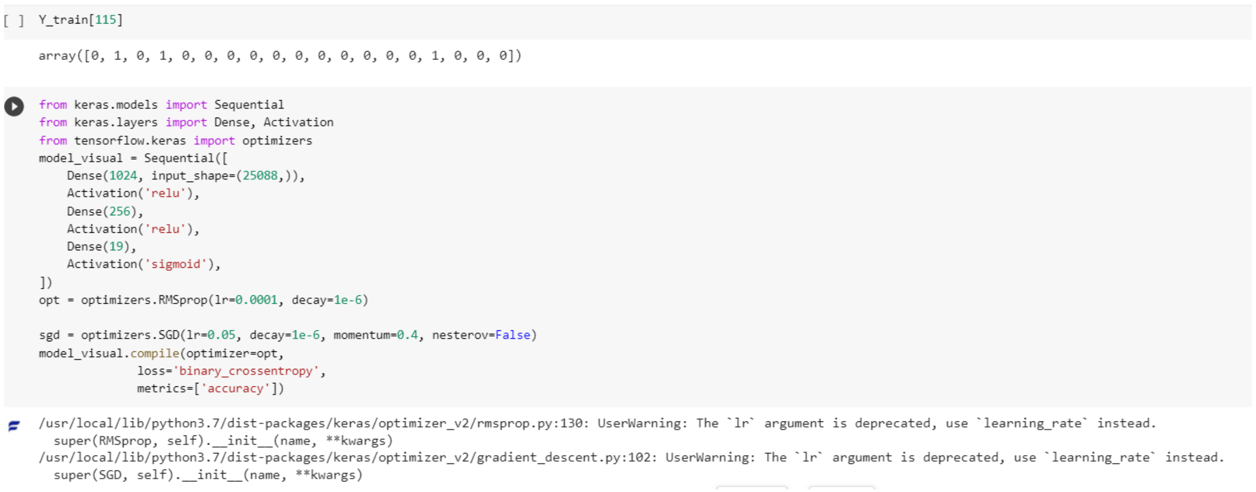
Binarized Y NumPy array contain binarized labels to the genre IDs of the 1553 movies



```
[ ]  visual_problem_data=(X,Y)
     f8=open('visual_problem_data_clean.pckl','wb')
     pickle.dump(visual_problem_data,f8)
     f8.close()

[ ]  f8=open('visual_problem_data_clean.pckl','rb')
     visual_features=pickle.load(f8)
     f8.close()

     (X,Y)=visual_features

[ ]  X.shape

     (1553, 25088)

[ ]  mask = np.random.rand(len(X)) < 0.8

[ ]  X_train=X[mask]
     X_test=X[~mask]
     Y_train=Y[mask]
     Y_test=Y[~mask]

[ ]  X_test.shape
     Y_test.shape

     (336, 19)
```

Creating our own koras neural network to use the VGG features and after that classify movie genres.

8. Training model after that, using the features we extracted from VGG net

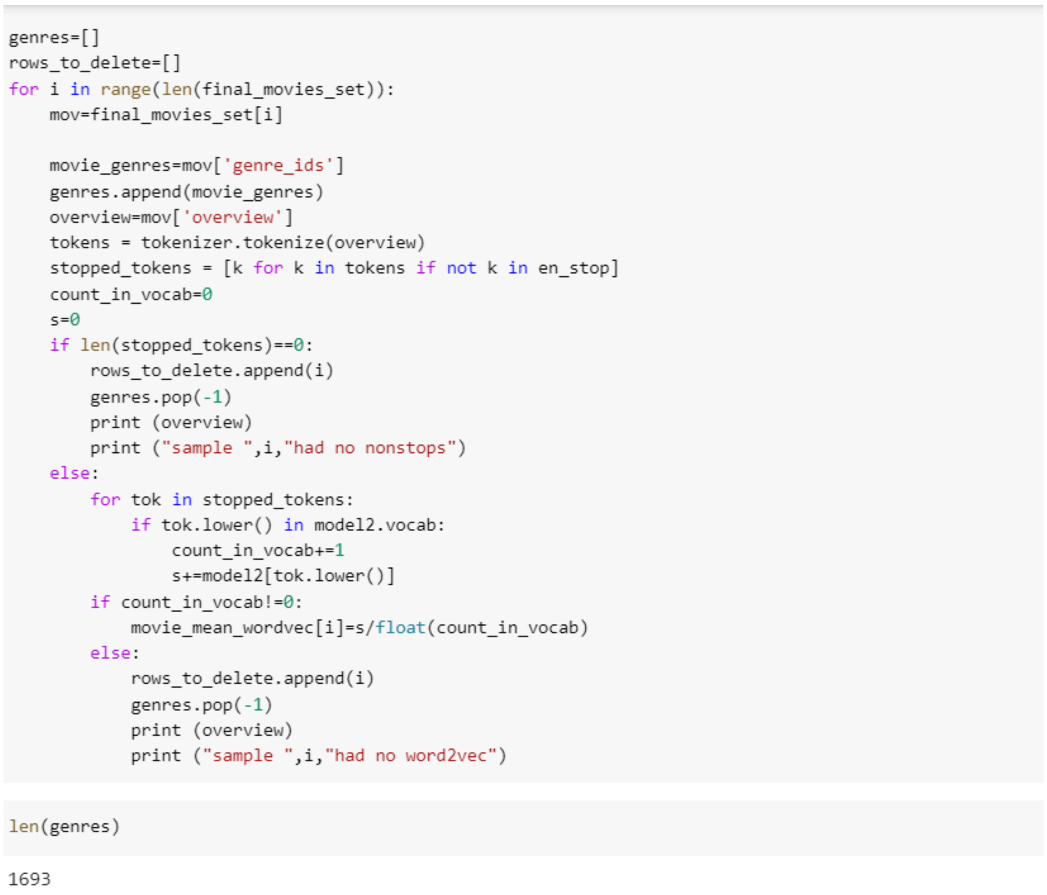


```
[ ] Y_train[115]

array([0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0])
```

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from tensorflow.keras import optimizers
model_visual = Sequential([
    Dense(1024, input_shape=(25088,)),
    Activation('relu'),
    Dense(256),
    Activation('relu'),
    Dense(19),
    Activation('sigmoid'),
])
opt = optimizers.RMSprop(lr=0.0001, decay=1e-6)

sgd = optimizers.SGD(lr=0.05, decay=1e-6, momentum=0.4, nesterov=False)
model_visual.compile(optimizer=opt,
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

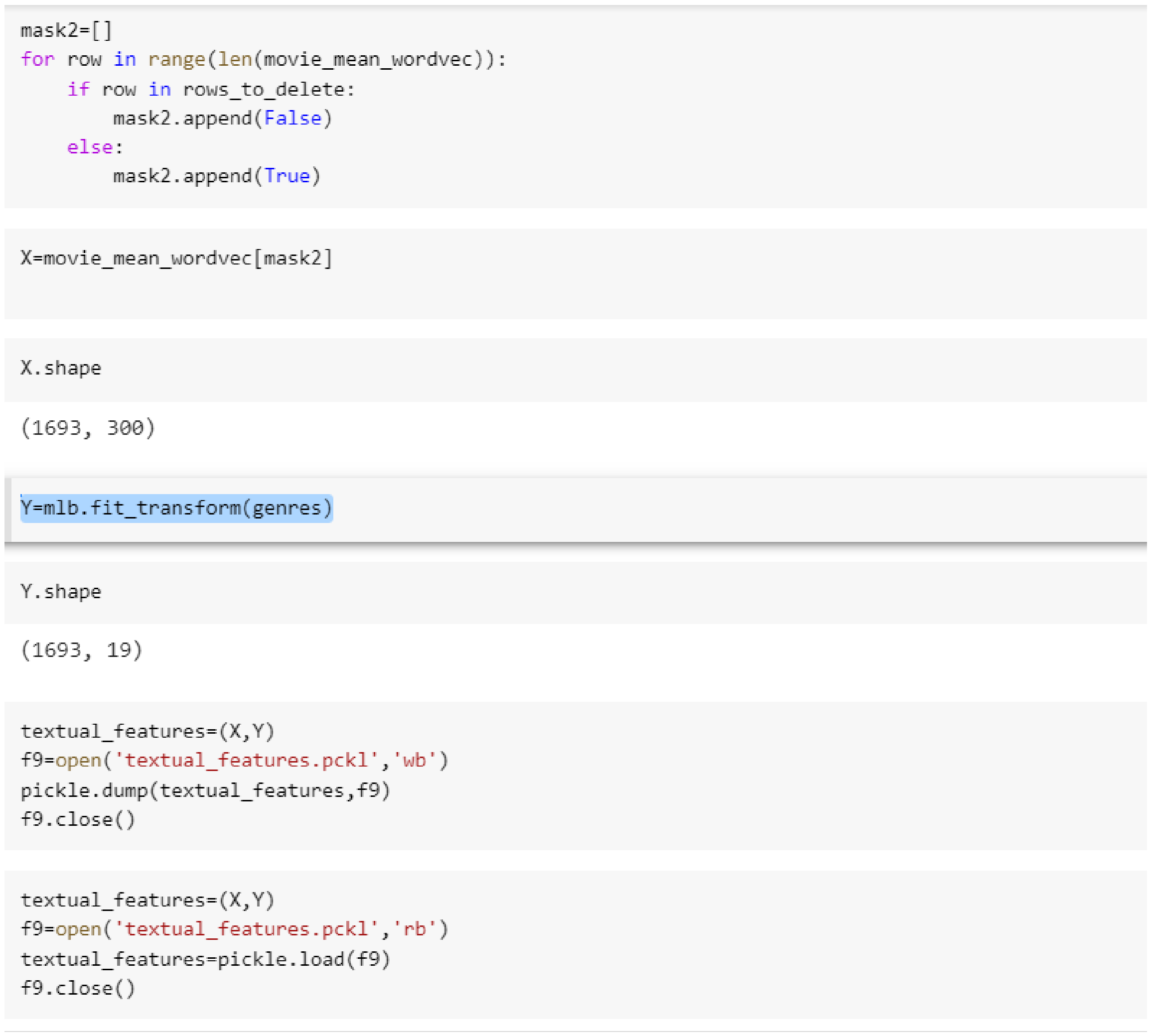
```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/rmsprop.py:130: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(RMSprop, self).__init__(name, **kwargs)
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/gradient_descent.py:102: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(SGD, self).__init__(name, **kwargs)
```

9. Solving matrix of X and Y



```
genres=[]
rows_to_delete=[]
for i in range(len(final_movies_set)):
    mov=final_movies_set[i]

    movie_genres=mov['genre_ids']
    genres.append(movie_genres)
    overview=mov['overview']
    tokens = tokenizer.tokenize(overview)
    stopped_tokens = [k for k in tokens if not k in en_stop]
    count_in_vocab=0
    s=0
    if len(stopped_tokens)==0:
        rows_to_delete.append(i)
        genres.pop(-1)
        print (overview)
        print ("sample ",i,"had no nonstops")
    else:
        for tok in stopped_tokens:
            if tok.lower() in model2.vocab:
                count_in_vocab+=1
                s+=model2[tok.lower()]
        if count_in_vocab!=0:
            movie_mean_wordvec[i]=s/float(count_in_vocab)
        else:
            rows_to_delete.append(i)
            genres.pop(-1)
            print (overview)
            print ("sample ",i,"had no word2vec")
```

```
len(genres)

1693
```



```
mask2=[]
for row in range(len(movie_mean_wordvec)):
    if row in rows_to_delete:
        mask2.append(False)
    else:
        mask2.append(True)
```

```
X=movie_mean_wordvec[mask2]
```

```
X.shape
```

```
(1693, 300)
```

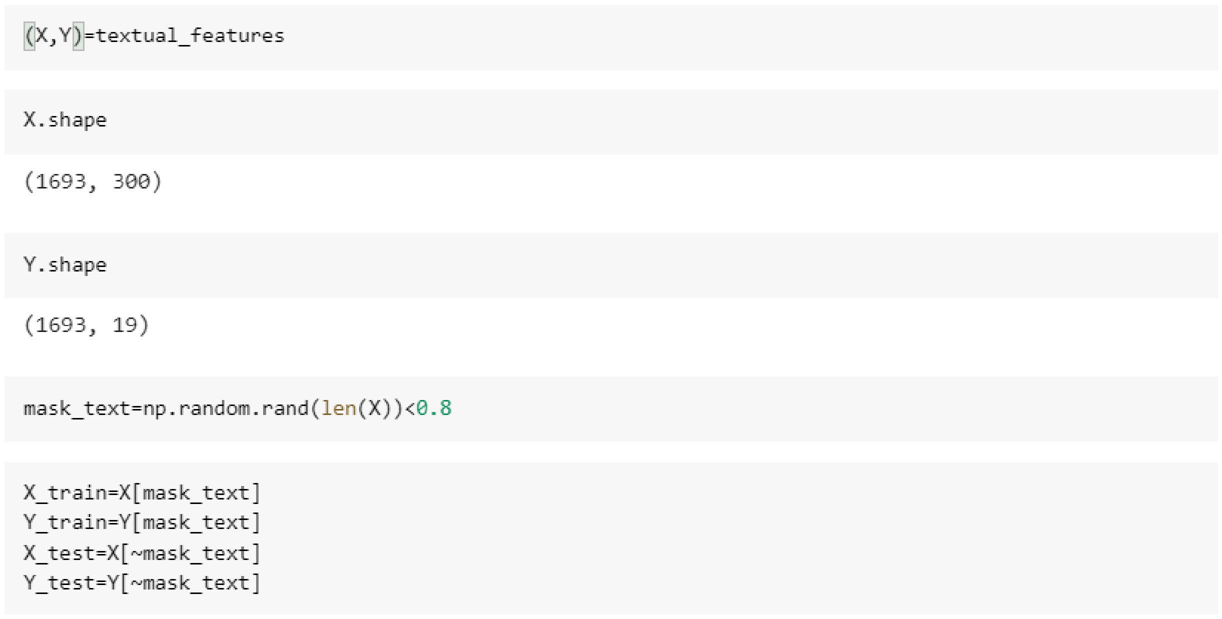
```
Y=mlb.fit_transform(genres)
```

```
Y.shape
```

```
(1693, 19)
```

```
textual_features=(X,Y)
f9=open('textual_features.pckl','wb')
pickle.dump(textual_features,f9)
f9.close()
```

```
textual_features=(X,Y)
f9=open('textual_features.pckl','rb')
textual_features=pickle.load(f9)
f9.close()
```



```
(X,Y)=textual_features
```

```
X.shape
```

```
(1693, 300)
```

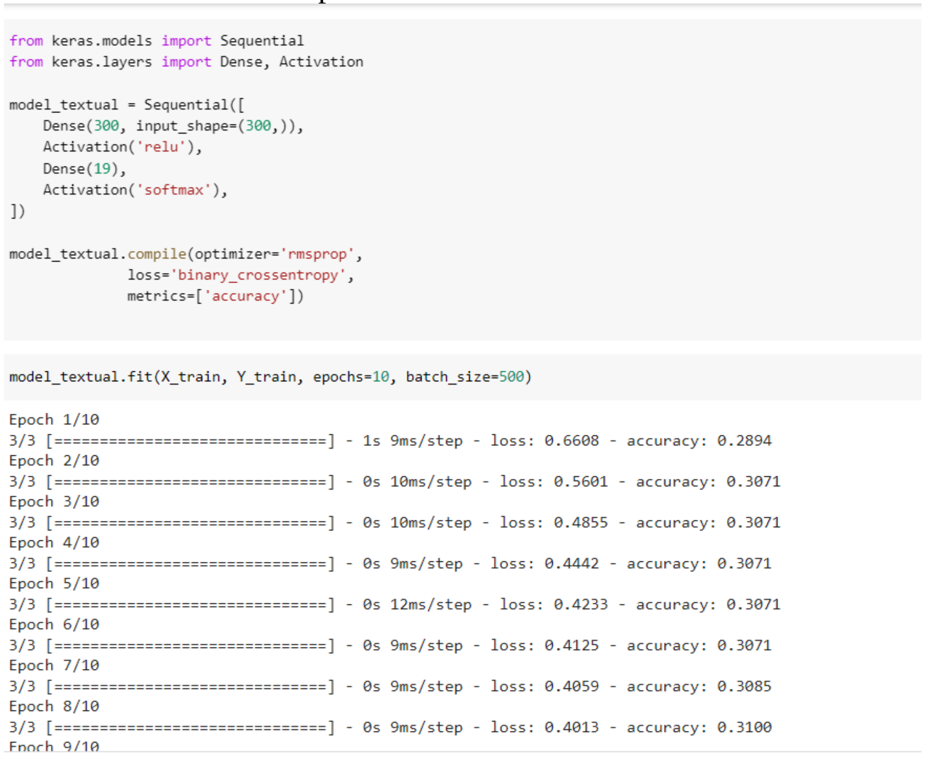
```
Y.shape
```

```
(1693, 19)
```

```
mask_text=np.random.rand(len(X))<0.8
```

```
X_train=X[mask_text]
Y_train=Y[mask_text]
X_test=X[~mask_text]
Y_test=Y[~mask_text]
```

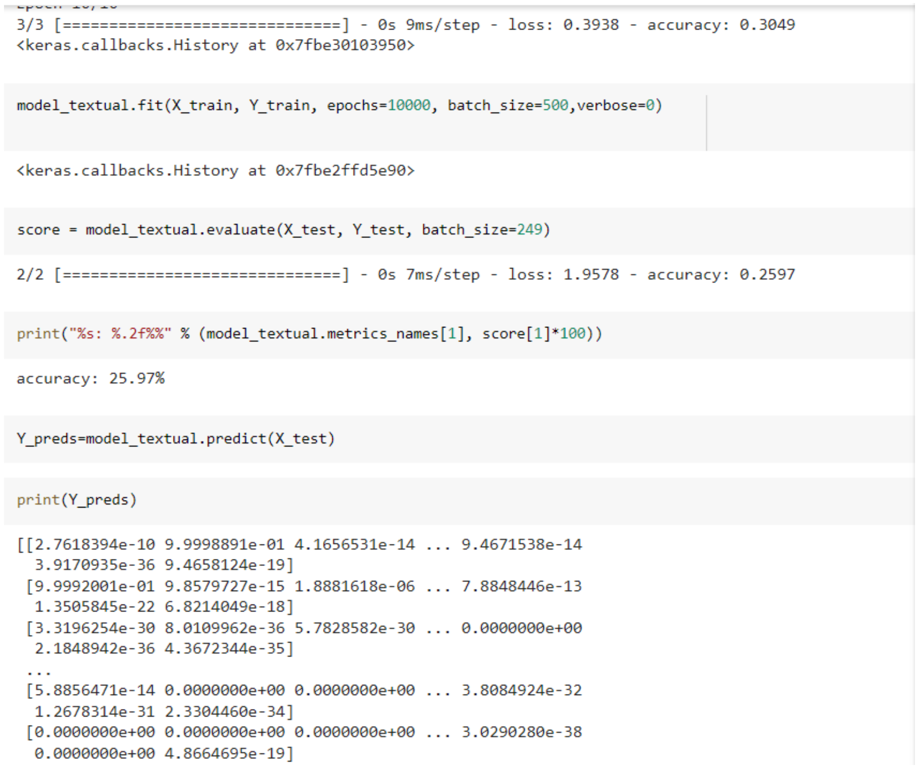
used simple architecture as before.



```
from keras.models import Sequential
from keras.layers import Dense, Activation

model_textual = Sequential([
    Dense(300, input_shape=(300,)),
    Activation('relu'),
    Dense(19),
    Activation('softmax'),
])

model_textual.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
model_textual.fit(X_train, Y_train, epochs=10, batch_size=500)
```

```
Epoch 1/10
3/3 [==============================] - 1s 9ms/step - loss: 0.6608 - accuracy: 0.2894
Epoch 2/10
3/3 [==============================] - 0s 10ms/step - loss: 0.5601 - accuracy: 0.3071
Epoch 3/10
3/3 [==============================] - 0s 10ms/step - loss: 0.4855 - accuracy: 0.3071
Epoch 4/10
3/3 [==============================] - 0s 9ms/step - loss: 0.4442 - accuracy: 0.3071
Epoch 5/10
3/3 [==============================] - 0s 12ms/step - loss: 0.4233 - accuracy: 0.3071
Epoch 6/10
3/3 [==============================] - 0s 9ms/step - loss: 0.4125 - accuracy: 0.3071
Epoch 7/10
3/3 [==============================] - 0s 9ms/step - loss: 0.4059 - accuracy: 0.3085
Epoch 8/10
3/3 [==============================] - 0s 9ms/step - loss: 0.4013 - accuracy: 0.3100
Epoch 9/10
```



```
3/3 [==============================] - 0s 9ms/step - loss: 0.3938 - accuracy: 0.3049
<keras.callbacks.History at 0x7fbe30103950>
```

```
model_textual.fit(X_train, Y_train, epochs=10000, batch_size=500,verbose=0)
```

```
<keras.callbacks.History at 0x7fbe2ffd5e90>
```

```
score = model_textual.evaluate(X_test, Y_test, batch_size=249)
```

```
2/2 [==============================] - 0s 7ms/step - loss: 1.9578 - accuracy: 0.2597
```

```
print("%s: %.2f%%" % (model_textual.metrics_names[1], score[1]*100))
```

```
accuracy: 25.97%
```

```
Y_preds=model_textual.predict(X_test)
```

```
print(Y_preds)
```

```
[[2.7618394e-10 9.9998891e-01 4.1656531e-14 ... 9.4671538e-14
  3.9170935e-36 9.4658124e-19]
 [9.9992001e-01 9.8579727e-15 1.8881618e-06 ... 7.8848446e-13
  1.3505845e-22 6.8214049e-18]
 [3.3196254e-30 8.0109962e-36 5.7828582e-30 ... 0.0000000e+00
  2.1848942e-36 4.3672344e-35]
 ...
 [5.8856471e-14 0.0000000e+00 0.0000000e+00 ... 3.8084924e-32
  1.2678314e-31 2.3304460e-34]
 [0.0000000e+00 0.0000000e+00 0.0000000e+00 ... 3.0290280e-38
  0.0000000e+00 4.8664695e-19]
```

```
genre_list.append(10769)
```

```
print ("Our predictions for the movies are - \n")
precs=[]
recs=[]
for i in range(len(Y_preds)):
    row=Y_preds[i]
    gt_genres=Y_test[i]
    gt_genre_names=[]
    for j in range(19):
        if gt_genres[j]==1:
            gt_genre_names.append(Genre_ID_to_name[genre_list[j]])
    top_3=np.argsort(row)[-3:]
    predicted_genres=[]
    for genre in top_3:
        predicted_genres.append(Genre_ID_to_name[genre_list[genre]])
    (precision,recall)=precision_recall(gt_genre_names,predicted_genres)
    precs.append(precision)
    recs.append(recall)
    if i%50==0:
        print ("Predicted: ",predicted_genres," Actual: ",gt_genre_names)
```

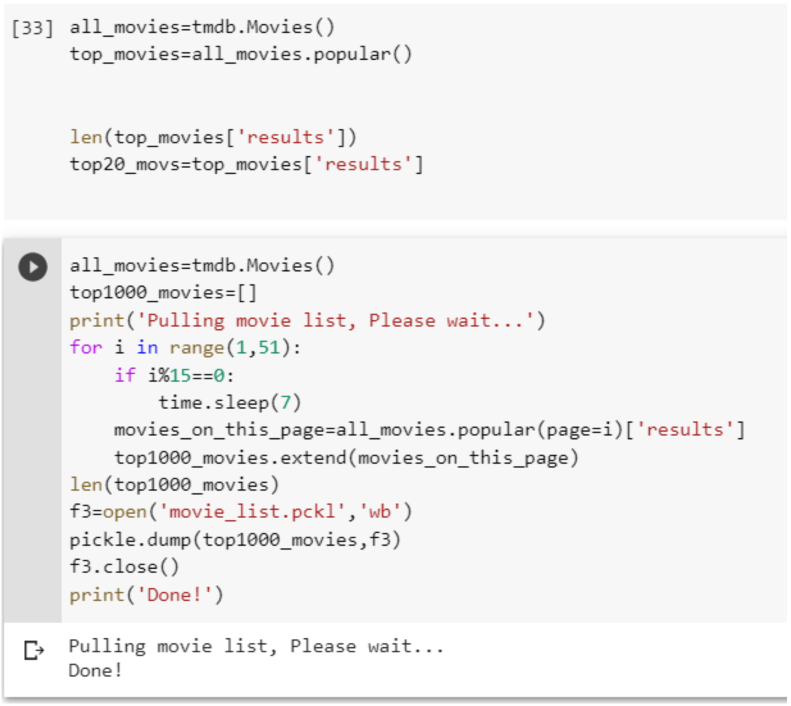
```
Our predictions for the movies are -

Predicted:  ['Adventure', 'Comedy', 'Fantasy']  Actual:  ['Adventure', 'Fantasy', 'Action']
Predicted:  ['Animation', 'Adventure', 'Action']  Actual:  ['Animation', 'Action', 'Science Fiction', 'Family']
Predicted:  ['Animation', 'Action', 'Adventure']  Actual:  ['Adventure', 'Fantasy', 'Action']
Predicted:  ['Drama', 'Action', 'Thriller']  Actual:  ['Action', 'Western', 'Science Fiction']
Predicted:  ['Mystery', 'Music', 'Documentary']  Actual:  ['Documentary']
Predicted:  ['Comedy', 'Crime', 'Music']  Actual:  ['Documentary', 'Music']
Predicted:  ['Family', 'Science Fiction', 'Comedy']  Actual:  ['Adventure', 'Comedy', 'Science Fiction', 'Family']
```

```
print (np.mean(np.asarray(precs)),np.mean(np.asarray(recs)))
```

```
0.5393034825870646 0.5807569296375267
```

Appendix 2

1.

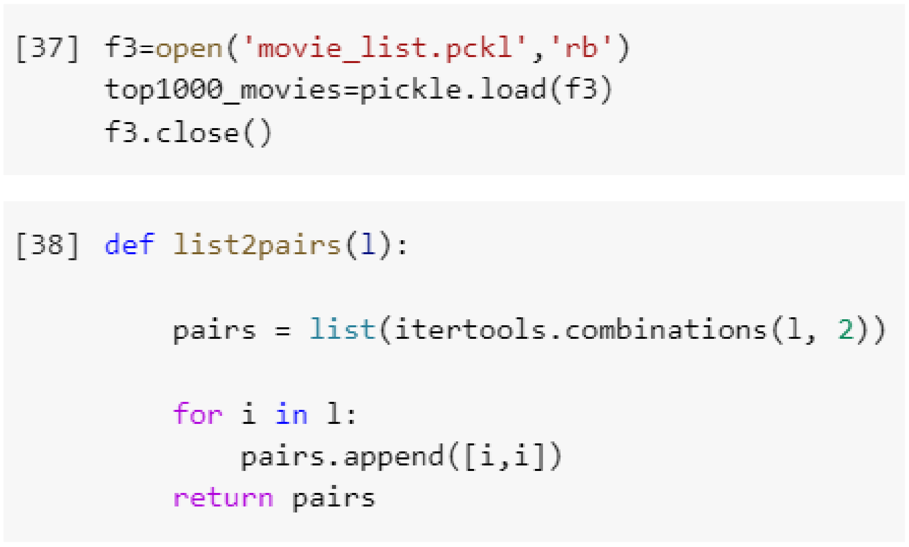


```
[33] all_movies=tmdb.Movies()
     top_movies=all_movies.popular()


     len(top_movies['results'])
     top20_movs=top_movies['results']
```

```
all_movies=tmdb.Movies()
top1000_movies=[]
print('Pulling movie list, Please wait...')
for i in range(1,51):
    if i%15==0:
        time.sleep(7)
    movies_on_this_page=all_movies.popular(page=i)['results']
    top1000_movies.extend(movies_on_this_page)
len(top1000_movies)
f3=open('movie_list.pckl','wb')
pickle.dump(top1000_movies,f3)
f3.close()
print('Done!')
```

```
Pulling movie list, Please wait...
Done!
```

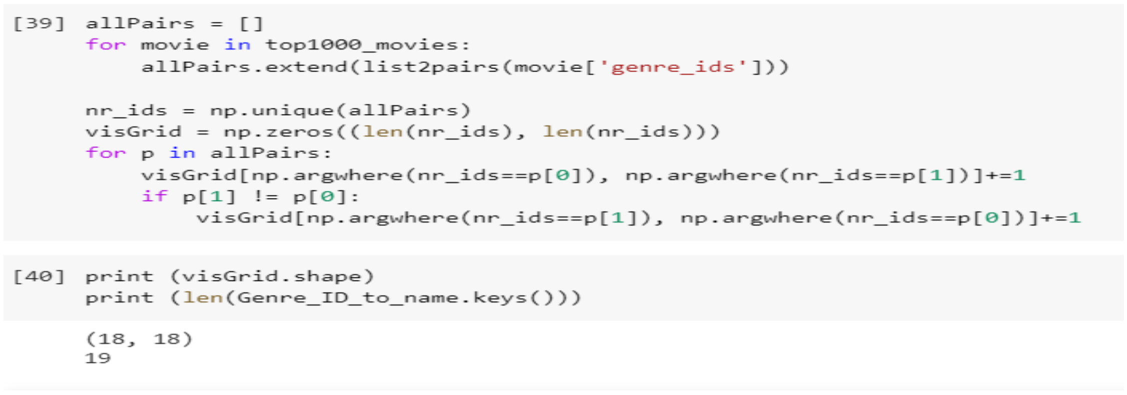
2.



```
[37] f3=open('movie_list.pckl','rb')
     top1000_movies=pickle.load(f3)
     f3.close()
```

```
[38] def list2pairs(l):

         pairs = list(itertools.combinations(l, 2))

         for i in l:
             pairs.append([i,i])
         return pairs
```

Pulling genres for each movie, and using above function to count occurrences of when two genres occurred together



```
[39] allPairs = []
     for movie in top1000_movies:
         allPairs.extend(list2pairs(movie['genre_ids']))

     nr_ids = np.unique(allPairs)
     visGrid = np.zeros((len(nr_ids), len(nr_ids)))
     for p in allPairs:
         visGrid[np.argwhere(nr_ids==p[0]), np.argwhere(nr_ids==p[1])]+=1
         if p[1] != p[0]:
             visGrid[np.argwhere(nr_ids==p[1]), np.argwhere(nr_ids==p[0])]+=1
```

```
[40] print (visGrid.shape)
     print (len(Genre_ID_to_name.keys()))
```

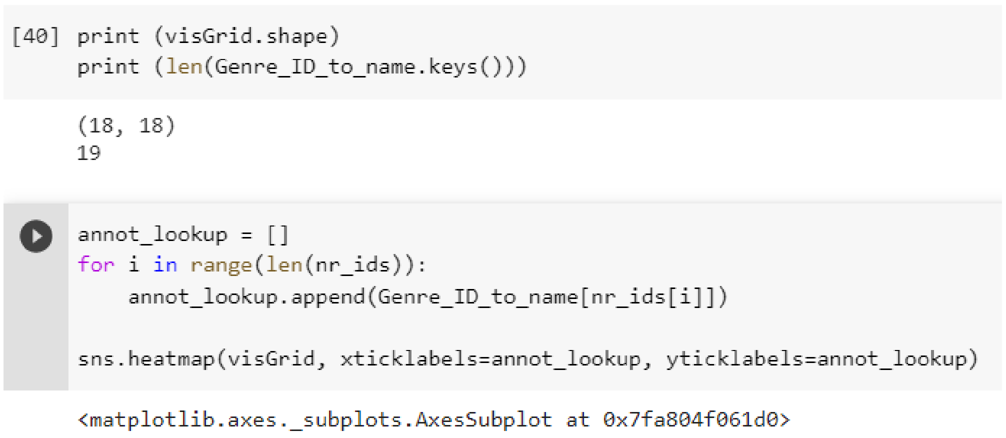
```
(18, 18)
19
```

19X19 structure, as shown below. 19 Genres.

This structure counts number of simultaneous occurrences of genres in same movie.

3.

4.

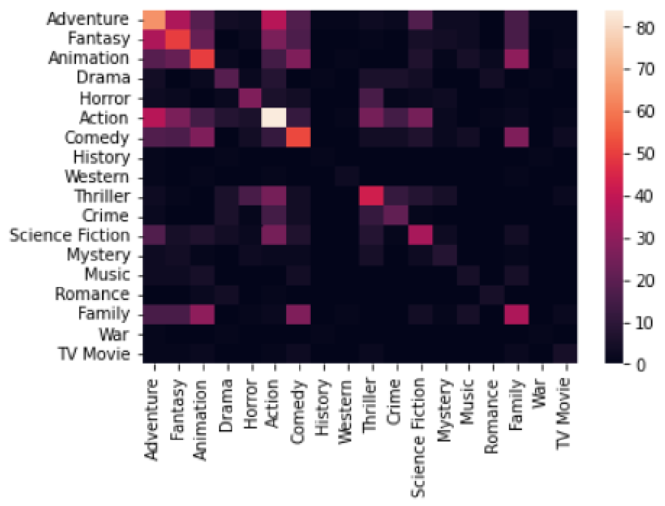


```
[40] print (visGrid.shape)
     print (len(Genre_ID_to_name.keys()))
```

```
(18, 18)
19
```

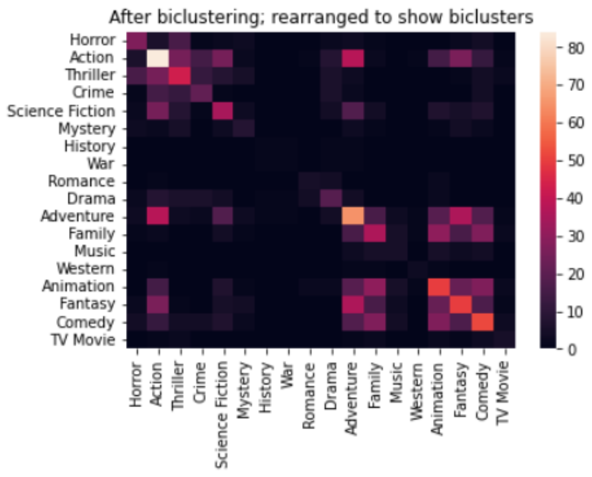
```
annot_lookup = []
for i in range(len(nr_ids)):
    annot_lookup.append(Genre_ID_to_name[nr_ids[i]])

sns.heatmap(visGrid, xticklabels=annot_lookup, yticklabels=annot_lookup)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa804f061d0>
```



```
[42] from sklearn.cluster import SpectralCoclustering
```

```
model = SpectralCoclustering(n_clusters=5)
model.fit(visGrid)

fit_data = visGrid[np.argsort(model.row_labels_)]
fit_data = fit_data[:, np.argsort(model.column_labels_)]

annot_lookup_sorted = []
for i in np.argsort(model.row_labels_):
    annot_lookup_sorted.append(Genre_ID_to_name[nr_ids[i]])

sns.heatmap(fit_data, xticklabels=annot_lookup_sorted, yticklabels=annot_lookup_sorted, annot=False)
plt.title("After biclustering; rearranged to show biclusters")

plt.show()
```



5.

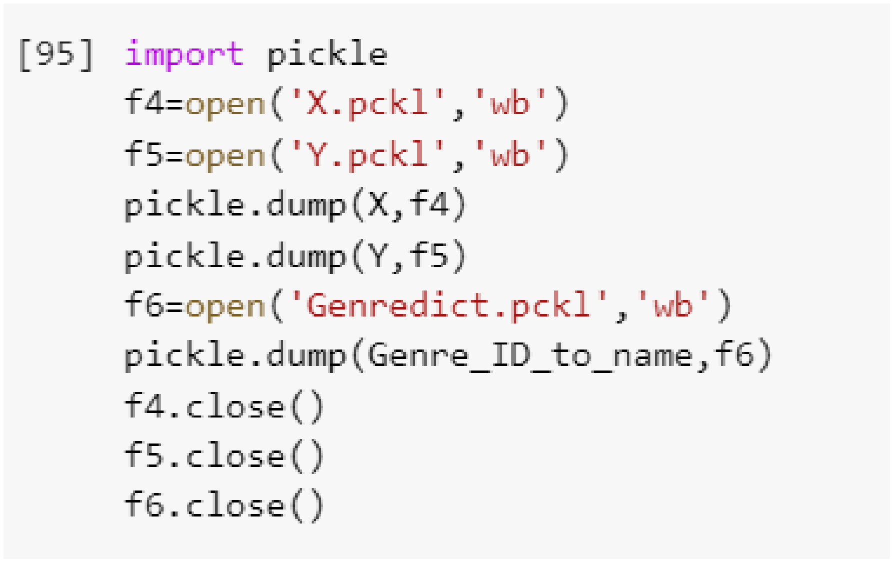
```
from sklearn.feature_extraction.text import CountVectorizer
import re
```

```
[85] content=[]
     for i in range(len(movies_with_overviews)):
         movie=movies_with_overviews[i]
         id=movie['id']
         overview=movie['overview']
         overview=overview.replace(',','')
         overview=overview.replace('.','')
         content.append(overview)
```

```
[86] print (content[0])
     print (len(content))
```

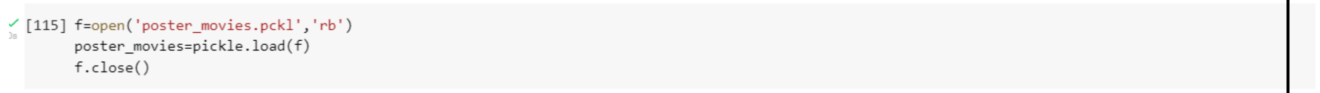
```
For Peter Parker life is busy Between taking out the bad guys as Spider-Man and spending time with the person
1600
```

6.

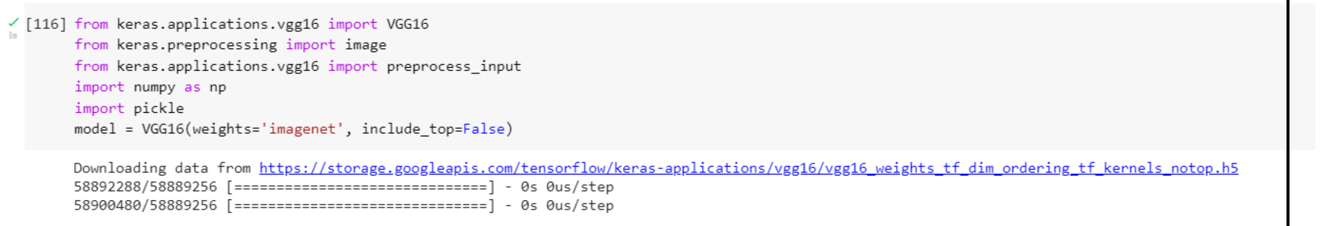


```
[95] import pickle
     f4=open('X.pckl','wb')
     f5=open('Y.pckl','wb')
     pickle.dump(X,f4)
     pickle.dump(Y,f5)
     f6=open('Genredict.pckl','wb')
     pickle.dump(Genre_ID_to_name,f6)
     f4.close()
     f5.close()
     f6.close()
```

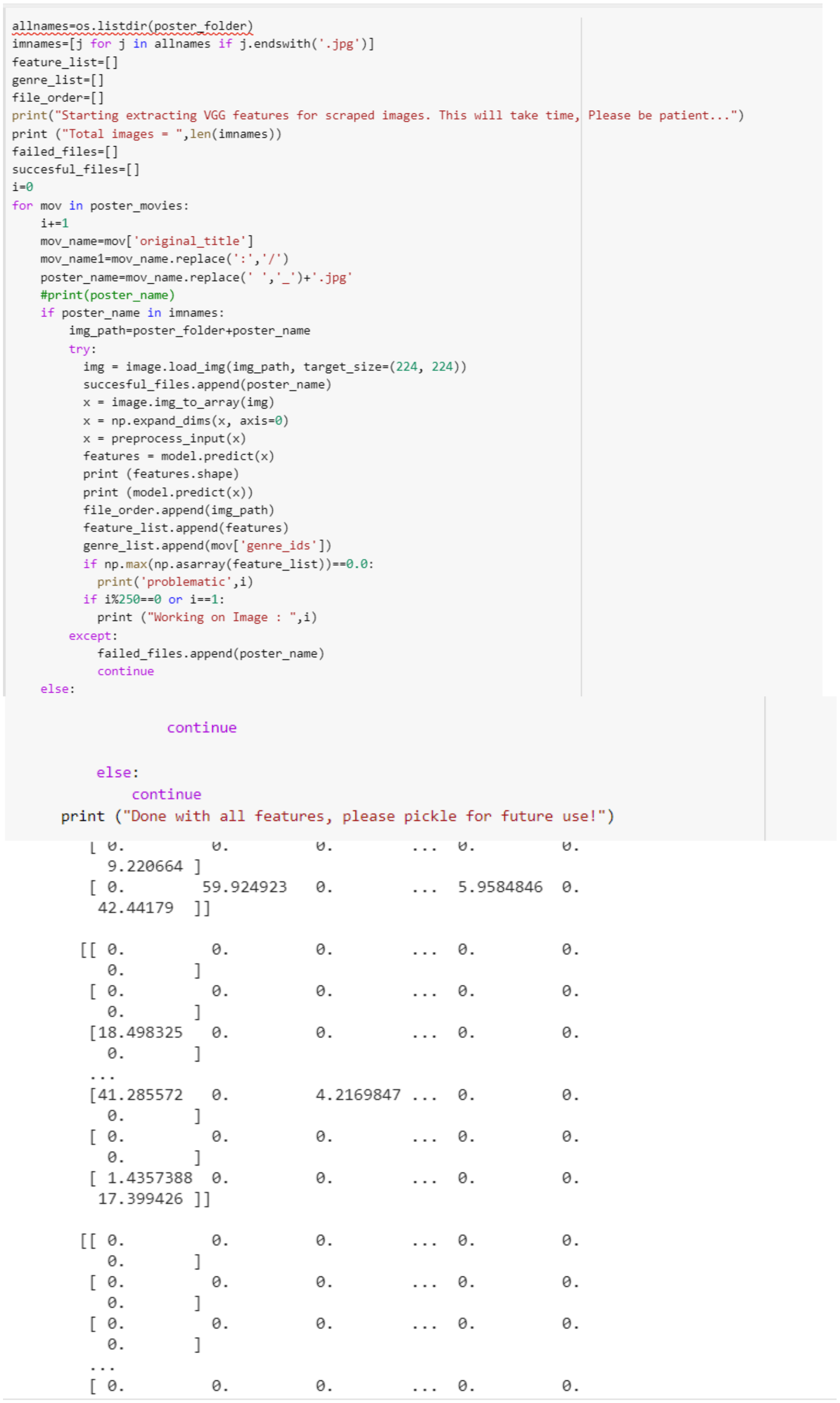
7.



```
[115] f=open('poster_movies.pckl','rb')
      poster_movies=pickle.load(f)
      f.close()
```



```
[116] from keras.applications.vgg16 import VGG16
      from keras.preprocessing import image
      from keras.applications.vgg16 import preprocess_input
      import numpy as np
      import pickle
      model = VGG16(weights='imagenet', include_top=False)

      Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
      58892288/58889256 [==============================] - 0s 0us/step
      58900480/58889256 [==============================] - 0s 0us/step
```



```
allnames=os.listdir(poster_folder)
imnames=[j for j in allnames if j.endswith('.jpg')]
feature_list=[]
genre_list=[]
file_order=[]
print("Starting extracting VGG features for scraped images. This will take time, Please be patient...")
print ("Total images = ",len(imnames))
failed_files=[]
succesful_files=[]
i=0
for mov in poster_movies:
    i+=1
    mov_name=mov['original_title']
    mov_name1=mov_name.replace(':','/')
    poster_name=mov_name.replace(' ','_')+'.jpg'
    #print(poster_name)
    if poster_name in imnames:
        img_path=poster_folder+poster_name
        try:
          img = image.load_img(img_path, target_size=(224, 224))
          succesful_files.append(poster_name)
          x = image.img_to_array(img)
          x = np.expand_dims(x, axis=0)
          x = preprocess_input(x)
          features = model.predict(x)
          print (features.shape)
          print (model.predict(x))
          file_order.append(img_path)
          feature_list.append(features)
          genre_list.append(mov['genre_ids'])
          if np.max(np.asarray(feature_list))==0.0:
            print('problematic',i)
          if i%250==0 or i==1:
            print ("Working on Image : ",i)
        except:
            failed_files.append(poster_name)
            continue
    else:

                continue

         else:
             continue
    print ("Done with all features, please pickle for future use!")
```

```
    [ 0.         0.         0.        ... 0.         0.
      9.220664  ]
    [ 0.         59.924923  0.        ... 5.9584846  0.
     42.44179   ]]

   [[ 0.         0.         0.        ... 0.         0.
      0.        ]
    [ 0.         0.         0.        ... 0.         0.
      0.        ]
    [18.498325   0.         0.        ... 0.         0.
      0.        ]
    ...
    [41.285572   0.         4.2169847 ... 0.         0.
      0.        ]
    [ 0.         0.         0.        ... 0.         0.
      0.        ]
    [ 1.4357388  0.         0.        ... 0.         0.
     17.399426 ]]

   [[ 0.         0.         0.        ... 0.         0.
      0.        ]
    [ 0.         0.         0.        ... 0.         0.
      0.        ]
    [ 0.         0.         0.        ... 0.         0.
      0.        ]
    ...
    [ 0.         0.         0.        ... 0.         0.
```