Categorization of YouTube Videos using Unsupervised Learning

Nitesh Gupta
Computer Science
Iowa State University
nkgupta@iastate.edu

Tanmay Gore
Computer Science
Iowa State University
tgore03@iastate.edu

Abstract

Categorization of web-based videos is an important task in video search and advertisement targeted applications. Unfortunately, the notion of a category varies depending upon the purpose, domain, and level of abstraction. We, therefore, use unsupervised learning algorithms to cluster similar videos together based on their description and metadata such as video statistics. The proposed method focuses on improving the intra-cluster similarity with a faster training time of the machine learning models. Preprocessing algorithm such as TF-IDF is implemented to improve the quality of data. PCA is applied to the resultant data to reduce the number of dimensions while preserving most of the information. Isomap is later used to visualize the structure of this data. Now with reasonable dimensions, we cluster the data using K-Means and Hierarchical Clustering. Since traditional accuracy measures cannot be applied to unsupervised learning, we determine the quality of clusters using Silhouette Coefficient.

Keywords: TF-IDF, PCA, YouTube, Isomap, K-Means, Agglomerative Hierarchical Clustering, Clustering, Dimensionality Reduction.

1. Introduction

Categorization of videos is an increasingly prominent area of research, rising with the increasing number of videos shared through online platforms such as YouTube. Its applications are of paramount importance to video recommendation and Advertisement targeted applications. Some recent applications are identifying fake, influential or misleading videos. However, the categorization of videos poses a great challenge. One major concern is that categories vary on the basis of their purpose, domain, and level of abstraction. Also, a video may belong to multiple categories making it difficult to define a fixed cluster size.

In this paper, we try to obtain an efficient clustering model inclined towards resolving above mentioned problems where the quality of a model is determined by the training time and similarity within clusters.

This project focuses on four main aspects – Data Preprocessing, Data Visualization, Clustering and evaluating the quality of Clustering. The dataset obtained from Kaggle about trending youtube videos contains various features such as title, description, tags, and metadata such as likes, dislikes, comments, and views.

In the first step, we preprocess the data by converting the textual information to TF-IDF. This greatly increases the dimension, hence to perform machine learning in a reasonable amount of time, we reduce the dimensions using PCA. The visualization step enables us to identify the structure of the data. We use Isomap to further

reduce the dimensions to "2" so as to visualize the structure of the data on a 2-D plane. Based on this structure, we decided to use K-Means and Agglomerative Hierarchical Clustering to obtain the clusters. Once the clusters were obtained, we validated its quality using Silhouette Coefficient.

1.1. TF-IDF

TF-IDF stands for *term frequency-inverse document frequency*, and the TF-IDF weight is often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance of a word is proportional to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

The following are the equations for TF and IDF respectively:

$$TF(t) = \frac{Number\ of\ times\ t\ occurs\ in\ a\ document}{Number\ of\ words\ in\ that\ document} \quad \ldots (1)$$

$$IDF(t) = log_e\left(\frac{Number\ of\ times\ t\ occurs\ in\ a\ document}{Number\ of\ words\ in\ that\ document}\right) \quad \ldots (2)$$

The TF-IDF is the product of equation (1) and (2).

1.2. Principle Component Analysis

Principal Component Analysis (PCA). PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that first principal component has the largest possible variance and each succeeding component has the highest possible variance under the constraint that it is orthogonal to the preceding components. A data point can have thousands of features. The running time for any machine learning algorithm depends on the number of data points in the dataset and the dimensionality of a data point. Through PCA it is possible to achieve the faster running time for the dataset by reducing the dimensions of a data point while preserving most of its value. Thus, PCA plays an important role in the situation when there is a need to perform various algorithms on a dataset to obtain the comparison results.

1.3. Isomap

Isomap is a nonlinear dimensionality reduction method. It is one of several widely used low-dimensional embedding methods. Isomap is used for computing a quasi-isometric, low-dimensional embedding of a set of high-dimensional data points. The algorithm provides a simple method for estimating the intrinsic geometry of a data manifold based on a rough estimate of each data point's neighbors on the manifold. Isomap is highly efficient and generally applicable to a broad range of data sources and dimensionalities.

1.4. K-Means

It is an unsupervised learning technique (i.e. no labels for the data are provided) which is used for clustering similar data into groups. The number of clusters k is the user-defined value. Each data point belongs to the cluster with closest mean.

1.5. Agglomerative Hierarchical Clustering

Hierarchical clustering (also called hierarchical cluster analysis or HCA) is a method of cluster analysis which seeks to build a hierarchy of

clusters. Agglomerative is a bottom-up approach where each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. The decision to merge cluster happens in greedy fashion and depends upon the metrics used to calculate the distance between clusters. The linkage criterion determines the distance between sets of observations as a function of the pairwise distances between observations. Single-linkage clustering happens in a bottom-up fashion where two clusters that contain the closest pair of elements not yet belonging to the same cluster are merged. In complete-linkage clustering, the link between two clusters contains all element pairs, and the distance between clusters equals the distance between those two elements (one in each cluster) that are farthest away from each other. The shortest of these links that remains at any step causes the fusion of the two clusters whose elements are involved. The method is also known as farthest neighbor clustering. In ward linkage, the criterion for choosing the pair of clusters to merge at each step is based on the optimal value of an objective function. If the objective function is the sum of square errors, then it is called ward's minimum variance method. Ward's minimum variance criterion minimizes the total within-cluster variance.

1.6. Silhouette Coefficient

Silhouette refers to a method of interpretation and validation of consistency within clusters of data. The technique provides a succinct graphical representation of how well each object lies within its cluster. The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters. The silhouette can be calculated with any distance metric, such as the Euclidean distance or the Manhattan distance.

2. Methodologies

To categorize the videos, we perform three major steps: (1) Preprocess Data (2) Clustering (3) Measure the quality of clusters. However, after preprocessing, we can visualize the data to better understand the structure. Each of the three steps focuses on getting a good quality clustering in terms of running time and intra-cluster similarity.

2.1. Preprocessing Data

The YouTube data contains various features such as title, description, tags, channel name, comments, views, likes, and dislikes. To process the textual features, we convert them to a numerical representation using TFIDF. We first assign a score for every word in video tags. This would enable us to have a vector representation of every feature that was obtained. Once the scores were obtained, a set containing every word was made. Let's call this set "*wordSet*". Every word in *wordSet* was assigned a unique index (*index word*). Next, a sparse zeros array containing dimensions [number of videos, length of *wordSet*] was generated. We iterated through the scores for every word for each video and added them to the specific index as determined by *index word*. This was done for every video in the sample space. Thus, the array contained the TF-IDF vectors for each video.

One benefit of TFIDF is that it retains the contextual information by assigning weighted scores to each word where higher weight is given to unique, meaningful words and lower weight to stop words. However, this significantly increases

the dimensions (50000 in our case). We, therefore, apply dimensionality reduction algorithm called PCA to reduce the dimensions. Since the matrix obtained after TFIDF is a sparse array matrix, PCA can greatly reduce the dimensions (3000 in our case) with 95% of the variance retained.

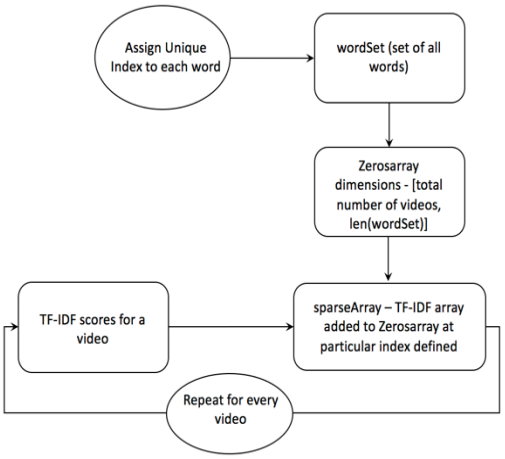

Figure 1 Process of TFIDF Algorithm

2.2. Visualizing the Data

Given the unknown nature of the data, it is crucial to visualize the data so as better determine the clustering algorithm to be used. However, the reduced dimensions (3000 in our case) are still too much for humans to visualize. We, therefore, apply another dimensionality reduction algorithm called Isomap to visualize the structure of the data. Isomap further reduces the dimensions to 2 by estimating the structural information of data.

2.3. Clustering the Data

Next, K Means and Agglomerative Hierarchical Clustering is applied to cluster the data. K Means is applied for different values of clusters ranging from 2 to 25. For each of the clusters, the model is trained 20 times independently with random seeds to overcome local minima.

2.4. Measure Quality of Clusters

Due to lack of labeled data, traditional accuracy measures cannot be applied to clustering. Therefore, we measure the quality of clusters obtained using Silhouette Coefficient. It determines the relationship between clusters and measures the similarity of data points within the cluster. For each of the model instance, we determine the silhouette coefficient.

3. Results and Discussions

Since the entire process is divided into 4 logical steps, namely Data Preprocessing, Data Visualization, Clustering and Quality Evaluation we will be discussing the results obtained in each of the 4 steps.

3.1. Data Preprocessing

Our video dataset contained information for around 4546 YouTube videos. To convert the textual features to numerical format, TFIDF was applied which converted the original 20 dimensions to 50000 dimensions. Since IDF was applied during the conversion, contextual meaning of data was retained using weighted scores.

To reduce the training time of machine learning algorithms, dimensionality reduction algorithm named PCA was applied. Due to the sparse data obtained from TFIDF, PCA reduced the dimensions from 50000 to 3000 with 95% of the variance retained.

3.2. Data Visualization

To better understand the underlying structure of data, another dimensionality reduction algorithm named Isomap was applied. Since Isomap

reduces the dimensions by retaining the structural information of the data, we could visualize the data in 2 or 3 dimensions.

By Plotting the PCA reduced data and non-PCA reduced data using Isomap we could see that PCA helped to normalize the data ensuring better clustering.

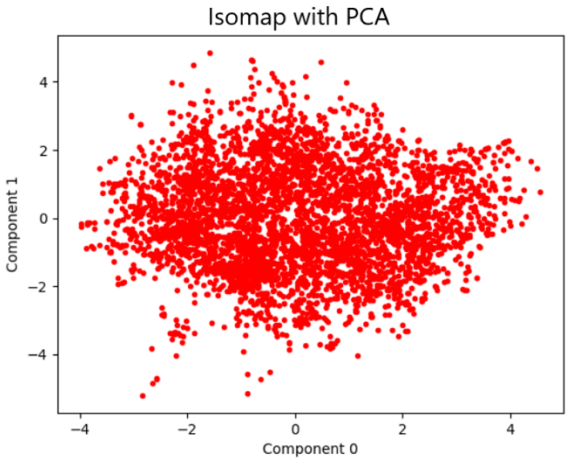

Figure 2: Structure of Data when PCA is applied

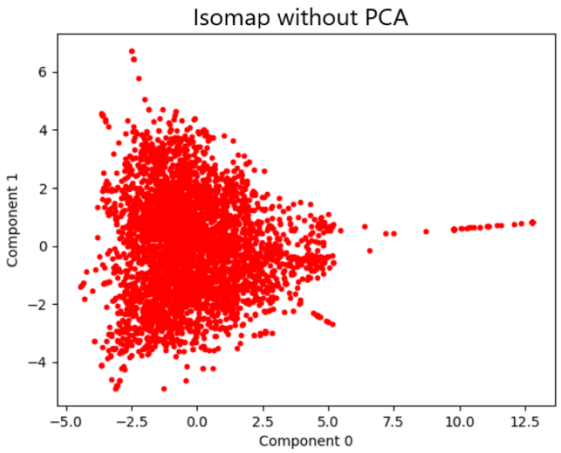

Figure 3: Structure of Data when PCA is not applied

3.3. Clustering

We performed clustering using K-Means and Agglomerative Hierarchical Clustering. Both the algorithms were performed for a different number of clusters. The K-Means model was trained for 20 epochs for each value.

The plots we got for cluster value 4 and 20 using K Means are as follows.

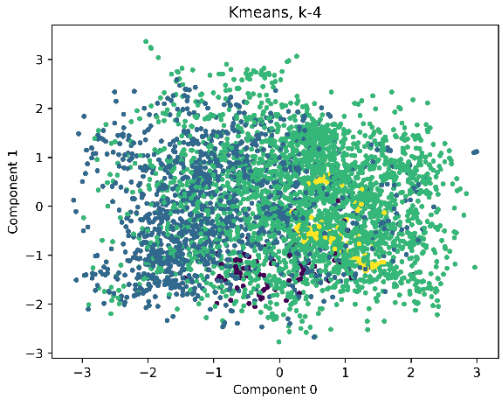

Figure 4: Clusters using K-Means for K=4

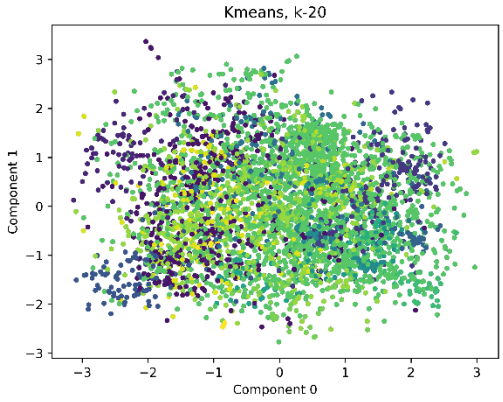

Figure 5: Clusters using K-Means for K=20

The plots we got using Hierarchical Clustering with linkage as average, complete and ward are as follows.

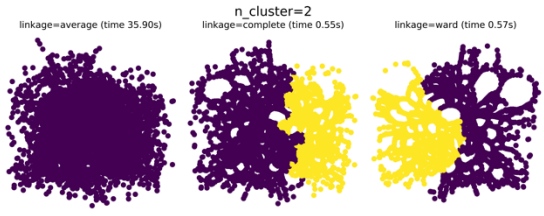

Figure 6: Hierarchical Clustering for clusters=2

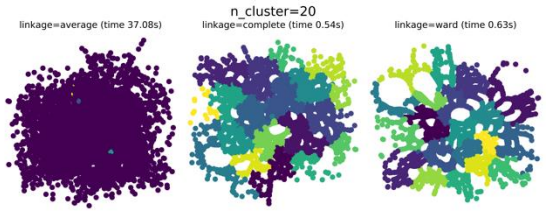

Figure 7: Hierarchical Clustering for clusters=20

3.4. Quality Evaluation

Since classical accuracy measures do not work on unsupervised learning due to lack of labels, we used Silhouette Coefficient. The Silhouette value obtained for different clusters using K-Means and its training time were as follows.

Clusters	Silhouette Coefficient	Training Time (sec)
2	-0.006	26.17
3	-0.003	28.08
4	-0.002	28.56
5	0.002	31.67
6	0.002	35.15
7	0.004	36.53
8	0.006	37.03
9	0.009	42.22
10	0.009	45.15
11	0.01	42.73
12	0.007	46.68
13	0.016	50.25
14	0.017	51.75
15	0.015	56.36
16	0.016	55.47
17	0.019	65.68
18	0.02	65.99
19	0.021	66.00
20	0.022	66.4
21	0.024	71.24
22	0.024	72.98
23	0.029	73.82
24	0.027	72.95
25	0.031	74.1

Table 1: Silhouette values using K-Means

K-Means	
K	20
Silhouette Coefficient	0.022
Number of Epochs	20
Training Time (sec)	65

Table 2: Best Parameter Values obtained for K-Means

Agglomerative Hierarchical Clustering	
Number of Clusters	20
Silhouette Coefficient	0.026
Training Time (sec)	83

Table 3: Best Parameter Values obtained for Hierarchical Clustering

4. Conclusion

This project focused primarily on obtaining a good clustering model for YouTube Videos. This project tries to obtain the best model in terms of intracluster similarity and Running Time by Preprocessing the data and using K-Means and Agglomerative Hierarchical Clustering. The quality of obtained clusters is validated using Silhouette Coefficient. We successfully obtained a clustering model using Agglomerative Hierarchical clustering with Silhouette Coefficient value of 0.026 which signifies the effective categorization of the videos. To improve the running time, we used PCA which helped in reducing the dimensions from 57000 to only 4000. TF-IDF not only helped to represent video description in numerical format but also helped to classify the videos based on meaningful words rather than useless words such as stop words. Isomap is used for visualization of data in a 2-D plane which helped in deciding the appropriate unsupervised learning algorithm. Thus, we were able to obtain a model with the very good amount of intra-cluster similarity, consistency and running time.

5. References

1. http://www.tfidf.com/ n.d.
2. https://en.wikipedia.org/wiki/Principal_component_analysis
3. https://en.wikipedia.org/wiki/Isomap
4. https://en.wikipedia.org/wiki/K-means_clustering
5. https://en.wikipedia.org/wiki/Hierarchichal_clustering
6. https://www.kaggle.com/datasets

6. Code

```
from io import open
import pandas as pd
import numpy as np
import time
import matplotlib.pyplot as plt
from sklearn.cluster import
KMeans
from sklearn.metrics import
silhouette_samples,
silhouette_score
from sklearn.manifold import
Isomap
from sklearn.cluster import
AgglomerativeClustering
from
sklearn.feature_extraction.text
import TfidfVectorizer
from sklearn.decomposition
import PCA
import pickle

def read_data(file,
label_values=None):
    #Read Data
    data = pd.read_csv(file)

    #Delete unnecessary columns
    columns=['video_id',
'last_trending_date',
'publish_date', 'publish_hour',
             'views', 'likes',
'dislikes', 'comment_count',
'comments_disabled',
             'ratings_disabled',
'tag_appeared_in_title_count',
'tag_appeared_in_title',
             'trend_day_count',
'trend.publish.diff',
'trend_tag_highest',

'trend_tag_total','subscriber','
channel_title']
    data = data.drop(columns,
axis=1)

    #Extract Labels from data
and convert data to Numpy Matrix
    labels =
np.array(data['category_id'].tol
ist())
    data =
data.as_matrix(columns =
['tags', 'title', 'description',
'tags_count'])
    #data =
data.as_matrix(columns =
['tags'])
    data = data.astype(str)

    #Replace "|" in Tags with "
"
    for i in range(len(data)):
        data[i,0] =
data[i,0].replace('|', ' ')

    #Merge multiple columns into
one column
    sep=" "
    for i in range(len(data)):
        data[i,0] =
sep.join(data[i])

    #Maps the values in label to
range(1, 16)
    y =
np.empty(shape=labels.shape,
dtype=int)
    if label_values is None:
        label_values =
set(labels)
        label_values =
list(label_values)

        for i in
range(len(labels)):
            y[i] =
label_values.index(labels[i])
    else:
        for i in
range(len(labels)):
            y[i] =
list(label_values).index(labels[
i])
```

```
    return data[:,0],y,
label_values

def store_array(filename,
array):
    np.save(filename, array)

def store_instance(filename,
instance):
    pickle.dump(instance,
open(filename, 'wb'))

def read_array(filename):
    array= np.load(filename)
    return array

def read_instance(filename):
    instance =
pickle.load(open(filename,
'rb'))
    return instance

def tf_idf(data):
    #TFIDF
    vectorizer =
TfidfVectorizer(stop_words='engl
ish')
    X =
vectorizer.fit_transform(data)
    X = X.todense()
    return X, vectorizer

def do_pca(X):
    pca = PCA(0.95)
    pca.fit(X)
    X = pca.transform(X)
    return X, pca

def do_isomap(X,n_comp):
    print("Performing isomap")
    imap =
Isomap(n_components=n_comp,
                   n_neighbors=5,
                   n_jobs=2,

neighbors_algorithm='auto')
    X_n = imap.fit_transform(X)
    return X_n

def plot2d_isomap(X, title,
y=None):
    plt.title(title)
    plt.xlabel('Component 0')
    plt.ylabel('Component 1')
    plt.legend()
    if y is not None:
        colors = [float(i)%
len(set(y)) for i in y]
        plt.scatter(X[:,0],
X[:,1], c=colors, marker='.')
    else:
        plt.scatter(X[:,0],
X[:,1], c='r', marker='.')
    #plt.show()

plt.savefig(""+str(title)+".png"
, dpi = 600)

def do_kmeans(X, X_n=None):
    if X_n is None:
        X_n = do_isomap(X,2)

    print("\nKMeans")
    k_range=[2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 19, 20, 21, 22, 23, 24,
25]
    for i in k_range:
        start_time =
time.clock()
        kmeans =
KMeans(n_clusters=i, n_init=20,
n_jobs=3)
        y =
kmeans.fit_predict(X)
        print("Time to build
KMeans model = ", time.clock()-
start_time)

        title="Kmeans, k-
"+str(i)
        plot2d_isomap(X_n,
title, y)
        print("For k=%d,
Silhouette Coefficient: %0.3f"
              % (i,
silhouette_score(X,
kmeans.labels_)))
    return kmeans

def
plot_isomap2d_AHC(X_n,model,inde
x,linkage,n_clusters,elapsed_tim
e):
    print("isomap 2D plot")
    plt.subplot(1, 3, index+1)
    plt.scatter(X_n[:, 0],
X_n[:, 1], c=model.labels_,
marker = '.')
    plt.title('linkage=%s (time
%.2fs)' % (linkage,
elapsed_time),
```

```
fontdict=dict(verticalalignment=
'top'))
    plt.axis('equal')
    plt.axis('off')

plt.subplots_adjust(bottom=0,
top=.89, wspace=0, left=0,
right=1)
    plt.suptitle('n_cluster=%i'
% (n_clusters), size=17)

def do_AHC(X, l, h, X_n=None):
    print("Aglomerative
Hierarchical clustering")
    if X_n is None:
        X_n = do_isomap(X,2)

    model =None
    for n_clusters in
range(l,h):
        plt.figure(figsize=(10,
4))
        for index, linkage in
enumerate(('average',
'complete', 'ward')):
            model =
AgglomerativeClustering(linkage=
linkage, n_clusters=n_clusters)
            t0 = time.time()
            model.fit(X)
            elapsed_time =
time.time() - t0
            print("Number of
Cluster=%d, linkage = %10s,
Silhouette Coefficient: %0.3f"%
(n_clusters, linkage,
silhouette_score(X,
model.labels_)))

            #plot 2d

plot_isomap2d_AHC(X_n,model,inde
x,linkage,n_clusters,elapsed_tim
e)

plt.savefig("num_clusters_"+str(
n_clusters)+".png",bbox_inches="
tight",dpi=600)
    plt.show()
    return model

def process_data(f):
    data, y, label_map =
read_data(f)
    X, tfidf = tf_idf(data)
    X, pca = do_pca(X)
    np.savetxt("X.csv", X,
delimiter=",")
    np.savetxt("y.csv", y,
delimiter=",")

    #Store the result
    store_array("X_array", X)
    store_array("y_array", y)

store_array("label_map",label_ma
p)

store_instance("tfidf.sav",tfidf
)
    store_instance("pca.sav",
pca)
    return X, y, label_map,
tfidf, pca

def build_models(f, X=None,
y=None, label_map=None,
tfidf=None, pca=None):
    if X is None:
        data, y, label_map =
read_data(f)
        X, tfidf = tf_idf(data)
        X, pca = do_pca(X)

    n_comp = 2
    X_n = do_isomap(X, n_comp)

    #KMeans
    start_time = time.clock()
    kmeans = do_kmeans(X, X_n)
    print("Time to build KMeans
model = ", time.clock()-
start_time)

    #Hierarchical Clustering
    start_time = time.clock()
    hc = do_AHC(X, 2, 4, X_n)
    print("Time to build
Hierarchical Clustering model =
", time.clock()-start_time)
    return label_map, tfidf,
pca, kmeans, hc

def main(file_train):
    #Preprocess data and store
it in file
    X, y, label_map, tfidf, pca
= process_data(file_train)
    print("Data Processed")

    #Read data from files
    X=read_array("X_array.npy")
    y=read_array("y_array.npy")
```

```
label_map=read_array("label_map.
npy")

tfidf=read_instance("tfidf.sav")
    pca=read_instance("pca.sav")
    print("Data read")

    # Building K-Means and
Agglomerative Hierarchical
Clustering(AHC) models
    label_map, tfidf, pca,
kmeans, hc =
build_models(file_train, X, y,
label_map, tfidf, pca)

file_train
='USvideos_modified.csv'

if __name__ == '__main__':
    main(file_train)
```

7. ReadMe

Steps to Run the Program:
1. Extract the zip folder
2. Open the ClassiTube.py python in python IDE or any other IDE
3. Run the program. (Note: It will a long time for the first time)

Steps to reduce the running time in the further runs:
1.Comment the line number 192 in ClassiTube.py (in main())
2.Now run the program as before.

The program stores the Data Transformation and model information in files and hence the short version read this stored information and trains the model on it.