563 Final Project: Classifying Trending Youtube Videos

Weikun Hu
Department of Applied Mathematics
University of Washington
Seattle, WA 98195
weikuh@uw.edu

Jun Song
Department of Applied Mathematics
University of Washington
Seattle, WA 98195
juns113@uw.edu

Huasong Zhang
Department of Applied Mathematics
University of Washington
Seattle, WA 98195
huasongz@uw.edu

Abstract

In this paper, we apply four machine learning methods to classify video types in "Trending YouTube Video Statistics" dataset [1]. These four methods are Support Vector Machine (SVM), K- Nearest Neighbors (KNN), Decision Tree and Random Forest. For pre-processing, we use several data-mining techniques including z-score normalization, imputation and feature engineering. At last, the accuracy of these four classification methods are compared.

1 Introduction and Overview

Trending helps viewers see what's happening on YouTube and in the world. Some trends are predictable, like a new song from a popular artist or a new movie trailer. Others are surprising, like a viral video. Trending aims to surface videos that a wide range of viewers will appreciate. According to Variety magazine, "To determine the year's top-trending videos, YouTube uses a combination of factors including measuring users interactions (number of views, shares, comments and likes). Note that they're not the most-viewed videos overall for the calendar year". Top performers on the YouTube trending list are music videos (such as the famously virile "Gangam Style"), celebrity and/or reality TV performances, and the random dude-with-a-camera viral videos that YouTube is well-known for [2].

The dataset we use is the US part of "Trending YouTube Video Statistics" [1]. This dataset includes several months (and counting) of data on daily trending YouTube videos from US. We pre-process the dataset with data-mining techniques such as imputation, z-score normalization and feature engineering. We then classify trending videos with low view based on 6 numerical features of the video, which are views, likes, dislikes, count, diff_days, and like_percentage. The classification methods we apply are SVM, KNN, decision tree and random forest. The highest classification accuracy among these methods is 50%. Given 16 categories of videos, the 50% accuracy from the model outperforms the baseline model with 6.25% accuracy.

2 Theoretical Background

2.1 Z-score Normalization

Standardization (or Z-score normalization) is the process of re-scaling the features so that they'll have the properties of a Gaussian distribution with $\mu = 0, \sigma = 1$, where μ is the mean and σ is the standard deviation from the mean; Standard scores (also called z scores) of the samples are calculated as follows [3]:

$$\vec{z} = \frac{\vec{x} - \text{mean}(\vec{x})}{\text{SD}(\vec{x})}$$

2.2 Support Vector Machine

The support vector machine algorithm is the optimization on the linear classifier that minimize the hinge loss for all misclassified points and maximize the size of the margin. The margin is defined as the distance of the closest point to the decision boundary.

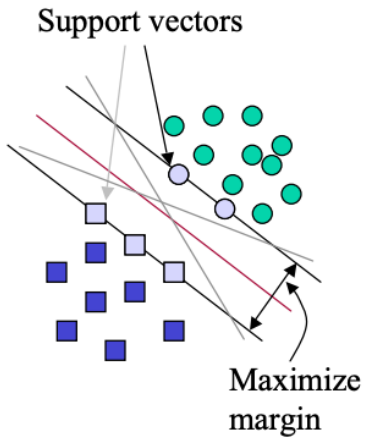

Figure 1: SVM Example

The support vector is the vectors from the decision boundary to the closest points, which shows in Figure 1. When the data in the original dimension is not linearly separable, a kernel function is applied to each data point to raise the dimension of the entire data set to make them become easily separable. The popular kernel functions are listed below

1. linear: <x,x'>
2. polynomial: $(\gamma < x, x' > +r)^d$
3. rbf: $exp(-\gamma||x - x'||^2)$
4. sigmoid: $(tanh(\gamma(x, x') + r))$

2.3 K Nearest Neighbor

KNN (K Nearest Neighbors) is one of many (supervised learning) algorithms used in data mining and machine learning, it's a classifier algorithm where the learning is based "how similar" is a data (a vector) from other. The KNN's steps are [4]:

1. Receive an unclassified data;
2. Measure the distance (Euclidian, Manhattan, Minkowski or Weighted) from the new data to all others data that is already classified;
3. Gets the K(K is a parameter that you define) smaller distances;
4. Check the list of classes had the shortest distance and count the amount of each class that appears;
5. Takes as correct class the class that appeared the most times;
6. Classifies the new data with the class that you took in step 5;

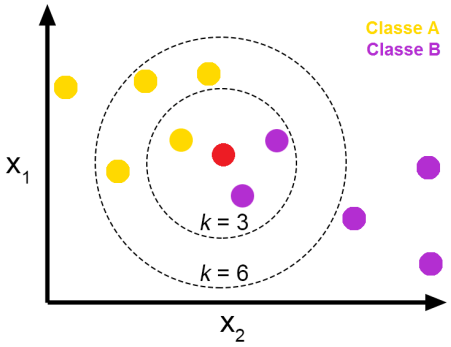

Figure 2: KNN Example

In Figure 2, when $K = 3$, among the 3 neighbor points, there are 2 from purple class and 1 from yellow class. So we will classify the test data point as purple. However, if we choose $K = 6$, among the 6 neighbor points, there are 4 from yellow class and 2 from purple class. So we will classify it as yellow.

2.4 Decision Tree

The decision tree is a hierarchical construct that looks for optimal ways to split the data in order to provide a robust classification and regression. It is the opposite of the unsupervised dendrogram hierarchical clustering previously demonstrated. In this case, our goal is not to move from bottom up in the clustering process, but from top down in order to create the best splits possible for classification. The fact that it is a supervised algorithm, which uses labeled data, allows us to split the data accordingly [5].

2.5 Random Forest

Random forests, or random decision forests, are an ensemble learning method for classification and regression. This is an important innovation since the decision trees created by splitting are generally not robust to different samples of the data. Thus one can generate two significantly different classification trees with two subsamples of the data. This presents significant challenges for cross-validation. In ensemble learning, a multitude of decision trees are constructed in the training process.

The random decision forests correct for a decision trees' habit of overfitting to their training set, thus providing a more robust framework for classification [5].

3 Algorithm Implementation and Development

3.1 Data Pre-processing

(1) *Handle Zero Values*:
We notice that there are some observations with 0 values in likes, dislikes and comment_count columns. We would like handle the 0 values before building the classification model. If all three columns are zero, we simply remove the observation. If one of variables has zero value, we treat it as a missing value and impute using KNN.

(2) *Generate New Features*:
We also generate new features to help to build the classification model. The features we add are:

- like_percentage: ratio of likes to likes + dislikes
- diff_days: difference in days between the video is publised and the video becomes trending

(3) *Normalize and Remove Outliers*:
We then normalize the numerical variables using z-score normalization. After normalization, we remove all the records where the numerical features are more than 3 standard deviations away.

(4) *Group Observations Based on Number of Views*:
The behavior of video with different number of views are very different, so we divide the observations into two groups. Low view group has videos with less than 1 million views and high view group which contains videos with more than 1 million views. In this assignment, we mainly focus on classifying low view trending videos.

3.2 Classification Methods

In this dataset, we have many attributes. However, some of them are categorical variables. In order to do the classification task, we only use the numerical features for classification, which are views, likes, dislikes, count, diff_days, and like_percentage.

(1) *SVM*:
We try SVM with several kernal functions such as RBF and linear kernel.

(2) *KNN*:
We not only use KNN to help pre-processing our dataset, we also apply it to classify the video categories. We calculate the accuracy rate based on different number of neighbors we take. Therefore, we build an array from 1 to 50, take each element as number of neighbors and loop through the entire array.

(3) *Decision Tree, Random Forest*:
For random forest, I firstly find the optimal minimum leaf size (minimum number of observations per tree leaf). Then I build the ensemble of bagged classification trees with this optimal minimum leaf size. To evaluate the performance of this random forest, I calculate the out-of-bag error (MATLAB function: *oobError*) and visualize the out-of-bag importance of the 6 numerical features (MATLAB function: *OOBPermutedVarDeltaError*).

4 Computational Results

4.1 Data Visualization

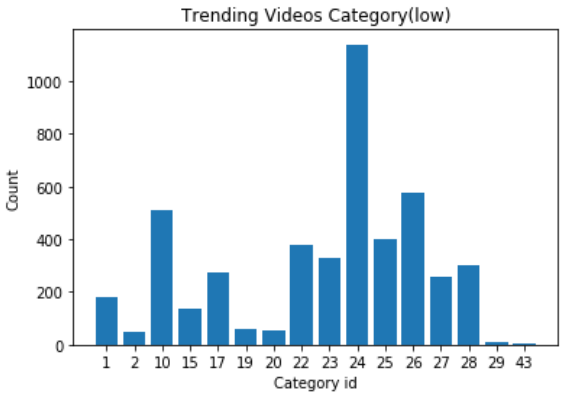

Figure 3: Trending Videos Categories

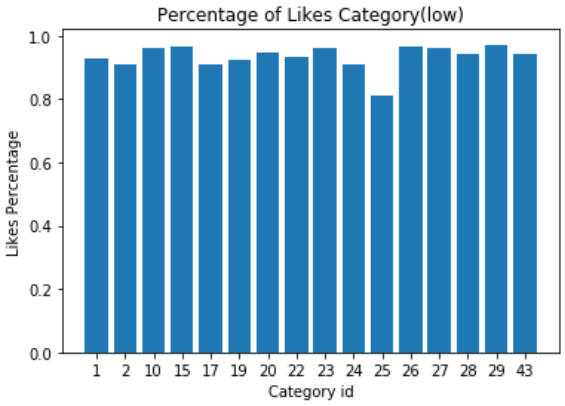

Figure 4: Like percentage for each category

Figure 3 shows the number of times that a category shows up in the low views group. As we can see from the plot that category 24 is the most popular ones among all the categories. And also, there are some categories that do not show up at all in low views group. There are possibilities that they are very popular and present in the high views group, or people are not interested in these groups at all.

However, when we look at the like percentage plot for each category as shown in Figure 4, we notice that even though category 24 is the most popular one watched by most people, the like percentage for this category is not the highest. And the like percentage can even be considered as low comparing to other categories.

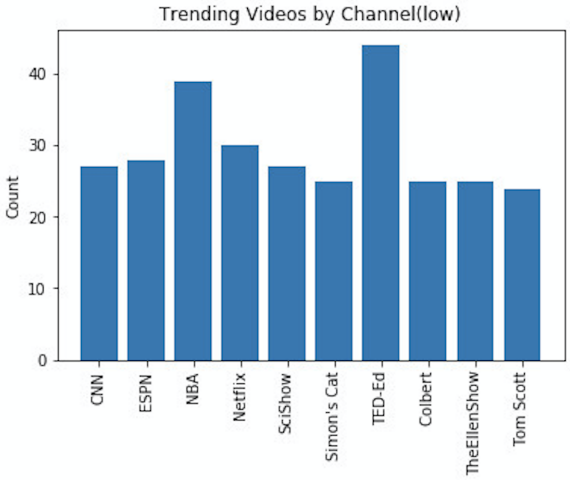

Figure 5: Trending Videos Channels

Figure 5 represents the top ten channels with most reviews. Ted-ed has been watched most frequently among all the channels.

4.2 KNN Results

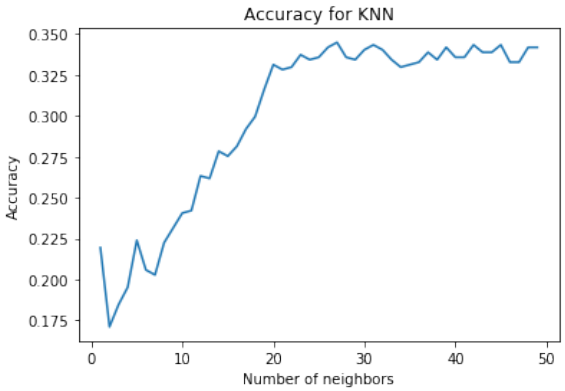

Figure 6: Accuracy from KNN

Figure 6 illustrates how the accuracy changes when the number of neighbors changes. As we can see from the plot that when number of neighbors is 1, the accuracy is about 20%. As we increase number of neighbors from 1 to 20, the classification accuracy also increases. When number of neighbors is larger than 20, the accuracy stops increasing and stabilizes around 34%.

4.3 Random Forest Results

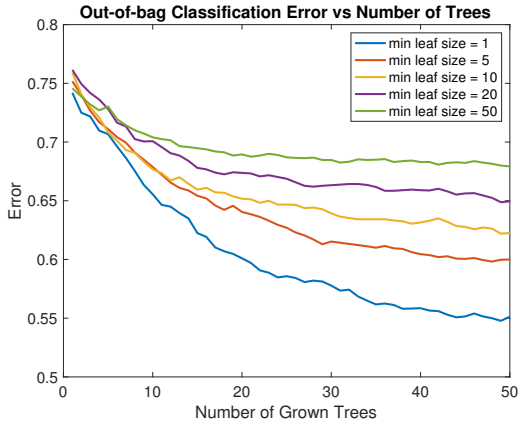

Figure 7: Classification Error vs Number of Trees (Different Minimum Leaf Size)

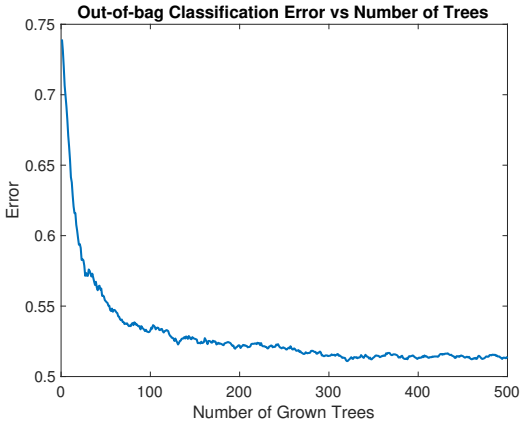

Figure 8: Out-of-bag Classification Error

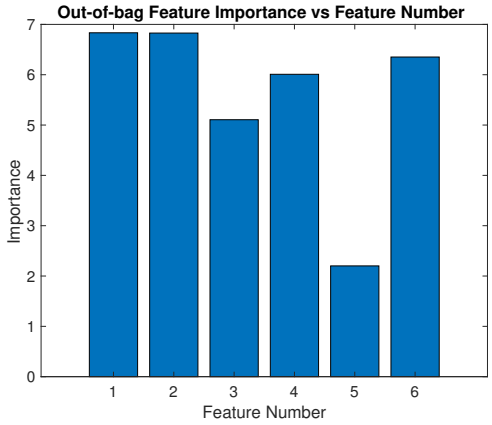

Figure 9: Out-of-bag Feature Importance, 1: views, 2: likes, 3: dislikes, 4: count, 5: diff_days, 6: like_percentage

From Figure 7, we observe that when minimum leaf size is 1, the model has the lowest classification error. Thus, optimal minimum leaf size is 1. The classification error under optimal minimum leaf size setting is in Figure 8, the out of bag classification error decreases to around 51% as number of grown trees increase to 300. Also, the out of bag feature importance in Figure 9 suggests that feature 1 (views) and feature 2 (likes) are of the highest importance.

Given the above findings, I train a random forest with 500 trees and minimum leaf size 1. The test classification accuracy is about 48%.

4.4 Other Methods

For decision tree, the classification accuracy is 21%. Using support vector machine algorithm, the prediction accuracy with linear kernel is 26%, while the accuracy with RBF kernel is 27%.

4.5 Comparison

Table 1: Classification Accuracy of Different Methods

Classification Method	Accuracy
SVM, linear	26%
SVM, RBF	27%
KNN	34%
Decision Tree	21%
Random Forest	48%

As in Table 1, Random Forest has the highest classification accuracy and KNN has the second highest accuracy.

5 Summary and Conclusions

In this project, we successfully pre-process and classify data of trending Youtube videos statistics from US. We obtain the highest classification accuracy 50% from random forest model. KNN achieves an accuracy of 35%, which is our second best model. The performance is satisfactory since the dataset has a very limited number of numerical features available for us to build feature space. Also, compared to the baseline model with 6.25% of accuracy, the accuracy of 50% is a significant improvement.

For future study, we can use bag-of-words or other natural language processing techniques to transform non-numeric features such as comment, tags, description, title to numerical frequencies. In this way, we can expand our feature space and potentially achieve a higher accuracy classifying trending Youtube videos. Besides, we only focus on the low view trending videos from US in this paper. Another interesting future direction is to apply our methods to high view trending Youtube videos from US and trending Youtube videos from other countries.

Appendix A Functions

- *generic random forests*: helper function to find optimal setting of random forest. This function evaluates classification accuracy under different minimum leaf size settings. Modified from `https://www.mathworks.com/matlabcentral/fileexchange/51985-simple-example-code-and-generic-function-for-random-forests?focused=3eb3a89f-d99a-c3ba-1936-1452fa2d4098&tab=function`.

Appendix B Codes

Data Pre-processing

```
########### Preparation
```{r}
# loading the required packages
library(plyr)
library(tidyverse)
library(wordcloud)
library(tm)
library(SnowballC)
library(lubridate)
library(ggcorrplot)
library(DMwR)
library(caret)
library(rpart)
library(rpart.plot)
library(pROC)
library(randomForest)
library(ipred)
library(caretEnsemble)
```


```{r}
```
```

```
# loading the dataset
youtube <- read.csv('../input/USvideos.csv')

# structure of the data
str(youtube)
```

########### Cleaning and Imputation

```{r}
# removing the thunmbnail link column
youtube <- youtube[,-12]

# setting category_id as a factor variable
youtube$category_id <- as.factor(youtube$category_id)

# setting all 0 values for likes, dislikes and comments as NA
youtube[, 9:11][youtube[, 9:11] == 0] <- NA

# counting the missing values
sapply(youtube, function(x) sum(is.na(x)))
```

```{r}
# removing the records where the likes and dislikes both are NA
youtube <- youtube %>%
  filter(!is.na(likes) & !is.na(dislikes))

# imputing the missing comment_count values using knn
knnOut <- round(knnImputation(youtube[,8:11], k = 10))

# inserting the imputed values into the original dataframe
youtube <- cbind(youtube[,1:10], knnOut[,4], youtube[,12:15])

# renaming the column
colnames(youtube)[11] <- 'comment_count'

# counting the missing values
sapply(youtube, function(x) sum(is.na(x)))
```

########### Generate New Features, Standardize
```{r}
# new feature that gives the ratio of likes and dislikes for a video
youtube$like_percentage <- youtube$likes/(youtube$dislikes+youtube$likes)

# converting the trending date into date format
youtube$trending_date <- ydm(youtube$trending_date)

# getting the publish date for the trending video
```

```
youtube$publish_date <- ymd(substr(youtube$publish_time, start = 1,
                                                    stop = 10))

# calculating the difference between the video being published and  trending
youtube$diff_days <- youtube$trending_date-youtube$publish_date

# converting diff_days into a numerical feature
youtube$diff_days <- as.numeric(youtube$diff_days)

# function to narmalize the numerical features
z_normalize <- function(x) {
  return ((x - mean(x)) / sd(x))
}

# normalizing the numerical features
youtube_norm <- cbind(youtube[,1:7], lapply(youtube[,c(8:11, 18)],
                z_normalize), youtube[,12:17])

# summary of numerical features after normalizing
summary(youtube_norm[,8:12])
```

Outliers, remove records with features 3 std away
```{r}
# videos with less than one million views
low_youtube <- youtube %>%
  filter(views < 1000000)

# normalizing the numerical features
low_youtube_norm <- cbind(low_youtube[,1:7],
                lapply(low_youtube[,c(8:11, 18)], z_normalize),
                low_youtube[,12:17])

# removing the rows with sd greater than 3
low_youtube_norm <- low_youtube_norm %>%
  filter(likes < 3 & dislikes < 3 & comment_count < 3 & diff_days < 3)

# summary of numerical features after normalizing
summary(low_youtube_norm[,8:12])
```

```{r}
# videos with views between one and ten million
high_youtube <- youtube %>%
  filter(views > 1000000 & views < 10000000)

# normalizing the numerical features
high_youtube_norm <- cbind(high_youtube[,1:7],
                lapply(high_youtube[,c(8:11, 18)], z_normalize),
                high_youtube[,12:17])
```

```r
# removing the rows with sd greater than 3
high_youtube_norm <- high_youtube_norm %>%
  filter(likes < 3 & dislikes < 3 & comment_count < 3 & views < 3
  & diff_days < 3)

# summary of numerical features after normalizing
summary(high_youtube_norm[,8:12])

## Error:  attempt to use zero-length variable name
```

Random Forest

```
% Author: Jun Song
% Last revision: 12-June-2019

T = readtable('train_low.csv');
data = T{:,[9:13,18]};
label = cellfun(@str2num,T{:,6});

training_set = data(1:4000,:);
testing_set = data(4000:4661,:);
training_label = label(1:4000,:);
testing_label = label(4000:4661,:);

%%% SVM
svm_model = fitcecoc(training_set,training_label);
svm_prediction = predict(svm_model,testing_set);
wrong_predicts = find(svm_prediction ~= testing_label);
accuracy1 = (length(testing_label) - length(wrong_predicts))...
        /length(testing_label);

%%% Naive Bayes
nb_model = fitcnb(training_set,training_label);
nb_prediction = predict(nb_model,testing_set);
wrong_predicts = find(nb_prediction ~= testing_label);
accuracy2 = (length(testing_label) - length(wrong_predicts))...
        /length(testing_label);

%%% KNN
knn_model = fitcknn(training_set,training_label);
knn_prediction = predict(knn_model,testing_set);
wrong_predicts = find(knn_prediction ~= testing_label);
accuracy3 = (length(testing_label) - length(wrong_predicts))...
        /length(testing_label);

%%% Random Forests
BaggedEnsemble = generic_random_forests(training_set, training_label, ...
    500, 'classification');
rf_prediction = predict(BaggedEnsemble, testing_set);
rf_prediction = cellfun(@str2num,rf_prediction);
wrong_predicts = find(rf_prediction ~= testing_label);
accuracy4 = (length(testing_label) - length(wrong_predicts))...
        /length(testing_label);
```

```
function BaggedEnsemble = generic_random_forests(X,Y,iNumBags,str_method)
```

```
leaf = [1 5 10 20 50];
figure
for i=1:length(leaf)
    b = TreeBagger(50,X,Y,'Method',str_method,'OOBPred','On',...
            'MinLeafSize',leaf(i));
    plot(oobError(b),'LineWidth',2);
    hold on
end
xlabel 'Number of Grown Trees'
ylabel 'Error'
title('Out-of-bag Classification Error vs Number of Trees');
set(gca, 'FontSize',15);
legend({'min leaf size = 1' 'min leaf size = 5' 'min leaf size = 10' ...
    'min leaf size = 20' 'min leaf size = 50'}, 'Location','NorthEast')
hold off

min_leaf_size = 1


%%%%%%%%%%%%%%%%%%%%  Plot out-of Bag Prediction Error  %%%%%%%%%%%%%%%%%%%%
BaggedEnsemble = TreeBagger(iNumBags,X,Y,'OOBPred','On','Method',str_method)
oobErrorBaggedEnsemble = oobError(BaggedEnsemble);
figID = figure;
plot(oobErrorBaggedEnsemble,'LineWidth',2);
title('Out-of-bag Classification Error vs Number of Trees');
xlabel 'Number of Grown Trees';
ylabel 'Error';
set(gca,'FontSize',15);
print(figID, '-dpdf', sprintf('randomforest_errorplot_%s.pdf', date));
oobPredict(BaggedEnsemble)

% view trees
% view(BaggedEnsemble.Trees{1}) % text description
% view(BaggedEnsemble.Trees{1},'mode','graph') % graphic description

%%%%%%%%%%%%%%%%%%%%%%%%  Estimate Feature Importance  %%%%%%%%%%%%%%%%%%%%%%%%
b = TreeBagger(iNumBags,X,Y,'Method',str_method,'OOBVarImp','On',...
    'MinLeafSize',min_leaf_size);

figure
plot(oobError(b),'LineWidth',2)
title('Out-of-bag MSE vs Number of Trees');
xlabel 'Number of Grown Trees';
ylabel 'Mean Squared Error';
set(gca,'FontSize',15);


figure
bar(b.OOBPermutedVarDeltaError)
set(gca,'FontSize',15);
xlabel 'Feature Number'
ylabel 'Importance'
title('Out-of-bag Feature Importance vs Feature Number');
idxvar = find(b.OOBPermutedVarDeltaError>0.7)
```

KNN, Decision Tree

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```
"""
Created on Tue May 28 21:25:07 2019
Modified by Jun Song on June 13 2019

@author: huasongzhang
"""

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from __future__ import division


train = pd.read_csv('train_low.csv')
(xx,yy) = train.shape

y_train = np.array([train.category_id])[0:1, 0:4000]
x_train = np.array([train.views,train.likes,train.dislikes,train.comment_count,
train.diff_days,train.like_percentage])[0:6, 0:4000]

y_test = np.array([test.category_id])[0:1, 4000:xx]
x_test = np.array([test.views,test.likes,test.dislikes,test.comment_count,
test.diff_days,test.like_percentage])[0:6, 4000:xx]

# plot the frequency appearances for each category(low)
categories_freq = train['category_id'].value_counts()
categories_freq.sort_index(inplace=True)
frequency = categories_freq.values
categories = categories_freq.index.values
plt.bar(list(map(str,categories)), frequency)
plt.xlabel('Category id')
plt.ylabel('Count')
plt.title('Trending Videos Category(low)')
plt.show()

# plot the like percentage for each category
train_like_per = train[['category_id', 'like_percentage']].copy()
like_per = train_like_per.groupby(['category_id']).mean()
plt.bar(list(map(str,categories)), like_per.like_percentage.values)
plt.xlabel('Category id')
plt.ylabel('Likes Percentage')
plt.title('Percentage of Likes Category(low)')
plt.show()

# plot Trending Videos by Channel(low)
channel_freq = train['channel_title'].value_counts()
low_channel_freq = channel_freq[0:10]
low_channel_freq.sort_index(inplace=True)
frequency_chan = low_channel_freq.values
```

```
channels = low_channel_freq.index.values
plt.bar(list(map(str,channels))[0:10], frequency_chan[0:10])
plt.xticks(rotation='vertical')
plt.xlabel('Channel')
plt.ylabel('Count')
plt.title('Trending Videos by Channel(low)')
plt.show()

def accuracy(truth,prediction):
    n = len(truth)
    count = 0
    for i in range(n):
        if truth[i] == round(prediction[i]):
            count += 1
    return count/n

# knn
nn = np.array([i for i in range(1,50)])
Acc_knn = []
for i in range(len(nn)):
    a = nn[i]
    classifier = KNeighborsClassifier(n_neighbors=a)
    fit_knn = classifier.fit(np.transpose(x_train), np.ravel(np.transpose(y_train)))
    y_pred_knn = fit_knn.predict(np.transpose(x_test))
    acc_knn = accuracy(np.ravel(np.transpose(y_test)),y_pred_knn)
    Acc_knn.append(acc_knn)

plt.plot(nn,Acc_knn)
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.title('Accuracy for KNN')
plt.show()


# decision tree
classifier1 = tree.DecisionTreeClassifier()
fit_tree = classifier1.fit(np.transpose(x_train), np.ravel(np.transpose(y_train)))
y_pred_tree = fit_tree.predict(np.transpose(x_test))
acc_tree = accuracy(np.ravel(np.transpose(y_test)),y_pred_tree)
print acc_tree
```

SVM

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


#Read data
```

```
from sklearn import svm
from sklearn import feature_extraction
from sklearn.metrics import confusion_matrix

import pandas as pd
import numpy as np #np.trace

df = pd.read_csv('test_low.csv')
df = pd.DataFrame(data=df)
df_test = pd.read_csv('test_low.csv')
df_test = pd.DataFrame(data=df_test)


A = df.groupby(['category_id']).count()

A = df.groupby(['category_id'],as_index=True).size()
print(A)
len(A)

import scipy
from tabulate import tabulate



Y_train = df['category_id']
#Y_train.head()
X_train =df.drop(columns =['category_id'])
# views + likes + dislikes + comment_count + diff_days

X_train_refined = df[['views','likes','dislikes','comment_count','diff_days']]
X_train_refined.head()


# Test data set
Y_train = df['category_id']
#Y_train.head()
X_train =df.drop(columns =['category_id'])

# views + likes + dislikes + comment_count + diff_days

X_train_refined = df[['views','likes','dislikes','comment_count','diff_days']]
X_train_refined.head()


# Test data set
Y_test = df['category_id']
#Y_test.head()
X_test =df.drop(columns =['category_id'])

# views + likes + dislikes + comment_count + diff_days
```

```
X_test_refined = df[['views','likes','dislikes','comment_count','diff_days']]
X_test_refined.head()


clf = svm.SVC(gamma='scale', decision_function_shape='ovo')
clf.fit(X_train_refined, Y_train)

lin_clf = svm.LinearSVC()
lin_clf.fit(X_train_refined, Y_train)

Y_pred = clf.predict(X_test_refined)

summary = confusion_matrix(Y_test,Y_pred)
TT = np.trace(summary)
SS = np.sum(summary)
print(TT)
print(SS)
acc = TT/SS
print(acc)


Y_pred_lin  = lin_clf.predict(X_test_refined)
summary = confusion_matrix(Y_test,Y_pred_lin)
TT = np.trace(summary)
SS = np.sum(summary)
print(TT)
print(SS)
acc = TT/SS
print(acc)

rbf_svc = svm.SVC(kernel='rbf')
rbf_svc.fit(X_train_refined, Y_train)

Y_pred = rbf_svc.predict(X_test_refined)

summary = confusion_matrix(Y_test,Y_pred)
TT = np.trace(summary)
SS = np.sum(summary)
print(TT)
print(SS)
acc = TT/SS
print(acc)
```

References

[1] Mitchell Jolly. *Trending YouTube Video Statistics*. https://www.kaggle.com/datasnaek/youtube-new.

[2] Mitchell Jolly. *Trending YouTube Video Scraper*. https://github.com/DataSnaek/Trending-YouTube-Scraper.

[3] Zaid Alissa Almaliki. *Standardization VS Normalization*. https://medium.com/@zaidalissa/standardization-vs-normalization-da7a3a308c64.

[4] Italo José. *KNN (K-Nearest Neighbors)*. https://towardsdatascience.com/@italojs.

[5] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. DOI: 10.1017/9781108380690.