YouTubeTrends

November 10, 2019

1 1 - Understanding YouTube Trends and Strategy for Going Viral

YouTube (the world-famous video sharing website) maintains a list of the top trending videos on the platform. Trending videos are not personalized and show what viewers in each country or region find interesting. For some videos it is predictable to become trending like a new song from a popular interpreter, or a new movie trailer. For other videos, the trends can be surprising. Within this notebook, we will explore trending videos and their YouTube titles with Natural Language Processing to build a machine learning model for an app that rates a new video title on a trending scale.

1.0.1 Data set

To conduct the study, we used open data from Kaggle, which includes several months of collecting data on daily trending YouTube videos. The dataset contains information about many regions around the world, but we only consider the U.S. data, which has up to 200 listed trending videos per day. This dataset was collected using the data scraper available on Github.

1.1 1.1 - Problem Statement: What decides if a video will be trending on YouTube?

YouTube remains vague about the patterns used in the trending algorithms and claims that some "soft facts" decide about trending. YouTube website says, that trending videos:

- are appealing to a wide range of viewers
- are not misleading, clickbaity, or sensational
- capture the breadth of what's happening on YouTube and in the world
- showcase a diversity of creators
- ideally are surprising or novel.

According to YouTube, trending algorithm aims to balance these considerations. To achieve this goal, trending algorithm considers many factors, including, view count, how fast the video generates views (i.e. "temperature"), where views are coming from, including outside of YouTube and the age of the video. Source: YouTube Help Center.

But what catches user's eye is at least the small thumbnail and the title text below the video. And even if YouTube is saying, clickbaity videos are not supported, while scrolling through YouTube trends everyone will find a lot of sensational and clickbaity results on the list.

Hard number metrics such as views or likes have likely to play a role, however it is not easy to discover how they are balanced and how they influence the overall trending ratings.

1.2 1.2 - Metrics for viral videos and trending

Even if YouTube does not openly admit, how the trending videos be found on the list, we will check how user interactions with videos such as likes, views and comments influence the trending. Likes or dislikes produce immediate feedback, a lot of comments might be an evidence of having a large and engaged community. Getting many views drives the video up to the top of the trending list. In this notebook, we will be studying how the title of a video affects the trending, and we will lay grounds to develop an app, which check a new video title, if it is good enough for the video to become trending.

---

2 2 - Analysis - Data Exploration and Data Visualization

In first step we will explore our data, drop irrelevant columns which are not interesting for the analysis. Further, we will look into correlations in the dataset, and specifics of the trending videos. Afterwards, we will investigate and report statistics about the metrics: likes, dislikes, views and comments, and normalize the metics.

2.1 2.1 - Importing relevant libraries.

[1]:
```
import numpy as np
import pandas as pd
import re
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.multioutput import MultiOutputClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDRegressor
from sklearn.svm import SVR
from sklearn.externals import joblib
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/kasialukoszek/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/kasialukoszek/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/kasialukoszek/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
/Users/kasialukoszek/anaconda3/lib/python3.7/site-
packages/sklearn/externals/joblib/__init__.py:15: DeprecationWarning:
sklearn.externals.joblib is deprecated in 0.21 and will be removed in 0.23.
Please import this functionality directly from joblib, which can be installed
with: pip install joblib. If this warning is raised when loading pickled models,
you may need to re-serialize those models with scikit-learn 0.21+.
  warnings.warn(msg, category=DeprecationWarning)
```

2.2 2.2 - Loading and preprocessing data

We decide only to look into US video data. We drop several information such us ids, urls and dates not relevant for this notebook.

[2]:
```
df = pd.read_csv("USvideos.csv")
df = df.drop(columns =␣
 ↪["description","video_id","publish_time","trending_date","thumbnail_link",␣
 ↪"category_id"])
```

2.3 2.3 - Data Statistics

[3]:
```
# Check the first lines from the data
df.head()
```

[3]:

	title	channel_title	tags	views	likes	dislikes	comment_count	comments_disabled	ratings_disabled
0	WE WANT TO TALK ABOUT OUR MARRIAGE	CaseyNeistat	SHANtell martin	748374	57527	2966	15954	False	False
1	The Trump Presidency: Last Week Tonight with J...	LastWeekTonight	last week tonight trump presidency|"last week ...	2418783	97185	6146	12703	False	False
2	Racist Superman | Rudy Mancuso, King Bach & Le...	Rudy Mancuso	racist superman|"rudy"|"mancuso"|"king"|"bach"...	3191434	146033	5339	8181	False	False
3	Nickelback Lyrics: Real or Fake?	Good Mythical Morning	rhett and link|"gmm"|"good mythical morning"|"...	343168	10172	666	2146	False	False
4	I Dare You: GOING BALD!?	nigahiga	ryan|"higa"|"higatv"|"nigahiga"|"i dare you"|"...	2095731	132235				

```
4      1989          17518              False             False

   video_error_or_removed
0                   False
1                   False
2                   False
3                   False
4                   False
```

[4]:
```
# View the concise summary of the dataframe
df.info()
```

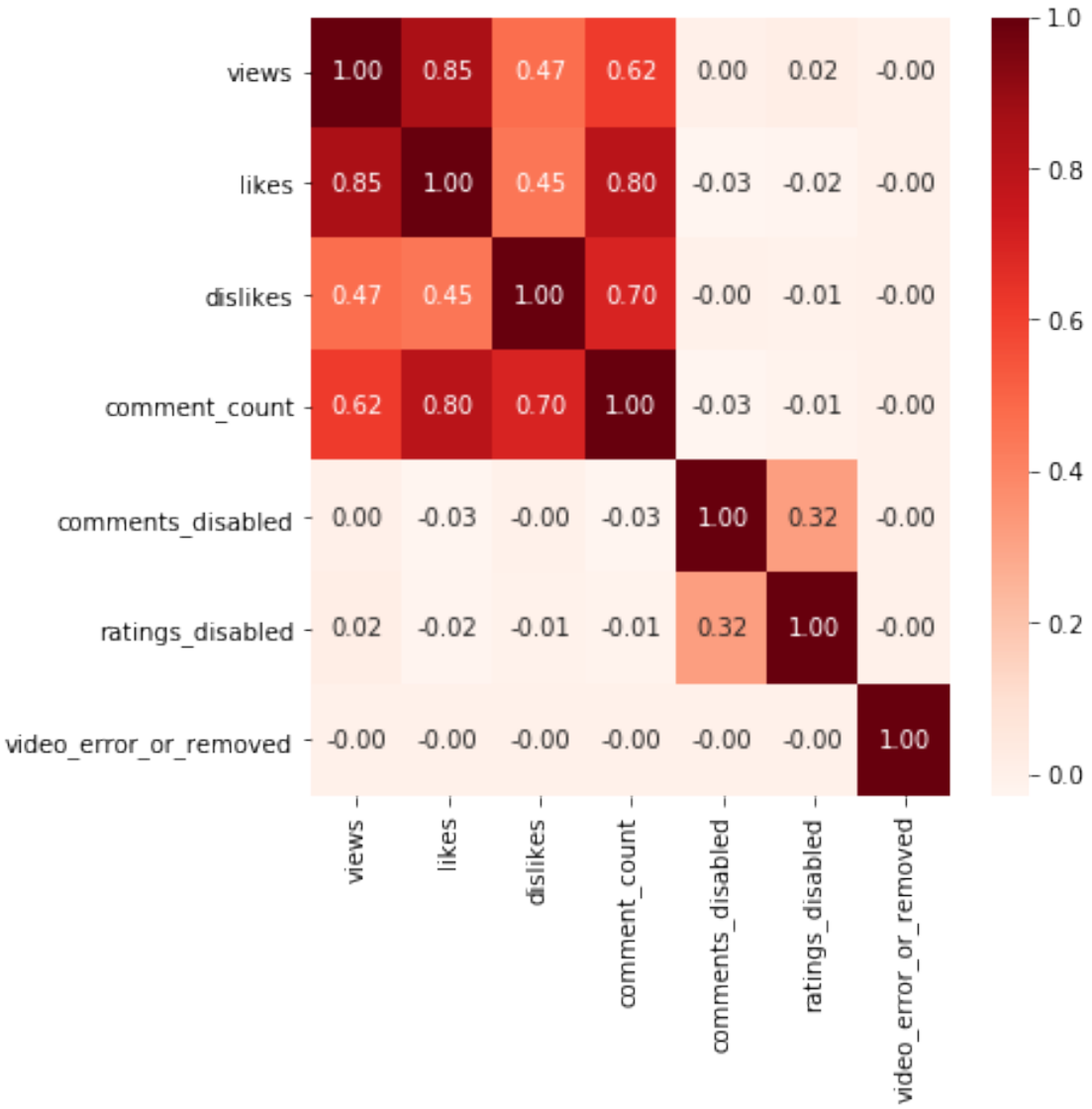
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40949 entries, 0 to 40948
Data columns (total 10 columns):
title                     40949 non-null object
channel_title             40949 non-null object
tags                      40949 non-null object
views                     40949 non-null int64
likes                     40949 non-null int64
dislikes                  40949 non-null int64
comment_count             40949 non-null int64
comments_disabled         40949 non-null bool
ratings_disabled          40949 non-null bool
video_error_or_removed    40949 non-null bool
dtypes: bool(3), int64(4), object(3)
memory usage: 2.3+ MB
```

2.4 2.4 - Correlations in the dataset

Looking at correlations ist one of the first ways to understand the data better. Even if a correlation between two variables does not automatically mean the causation between variables, it might gives hints for a further analysis.

In our correlation matrix we observe strong correlations between views, likes, dislikes an comment_count.

[5]:
```
plt.figure(figsize=(6,6))
cor = df.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds, fmt=".2f")
plt.show()
```



2.5 2.5 - Characteristics of Trending Videos

Here we investigate characteristics of the trending videos, to better understand, which videos

2.5.1 Top 10 YouTube Trending channels in the U.S.

[6]:
```
df["channel_title"].value_counts().head(10)
```

[6]:
```
ESPN                                    203
The Tonight Show Starring Jimmy Fallon  197
Vox                                     193
TheEllenShow                            193
Netflix                                 193
```

```
The Late Show with Stephen Colbert        187
Jimmy Kimmel Live                         186
Late Night with Seth Meyers               183
Screen Junkies                            182
NBA                                       181
Name: channel_title, dtype: int64
```

2.5.2 Top 10 Trending Tags in the U.S.

[7]:
```
def tokenize_tags(df):
    flatten_tags = df["tags"].iloc[:10].str.lower().str.cat(sep=' ')
    tags_cleaned = re.sub('[^A-Za-z]+', ' ', flatten_tags)
    tokens = word_tokenize(tags_cleaned)
    lemmatizer = WordNetLemmatizer()
    default_stopwords = set(stopwords.words("english"))
    tokens = [w for w in tokens if w not in default_stopwords]
    clean_tokens = []
    for tok in tokens:
        clean_tok = lemmatizer.lemmatize(tok).lower().strip()
        clean_tokens.append(clean_tok)
    return clean_tokens
```

[8]:
```
cleaned_data_title = tokenize_tags(df)
list = pd.Series(cleaned_data_title)
list.value_counts().head(10)
```

[8]:
```
gadget        8
nickelback    8
iphone        6
musical       5
mythical      5
x             5
trailer       5
link          4
barnum        4
lyric         4
dtype: int64
```

2.6 2.5 - Distribution of the metrics: likes, dislikes, views and comment count.

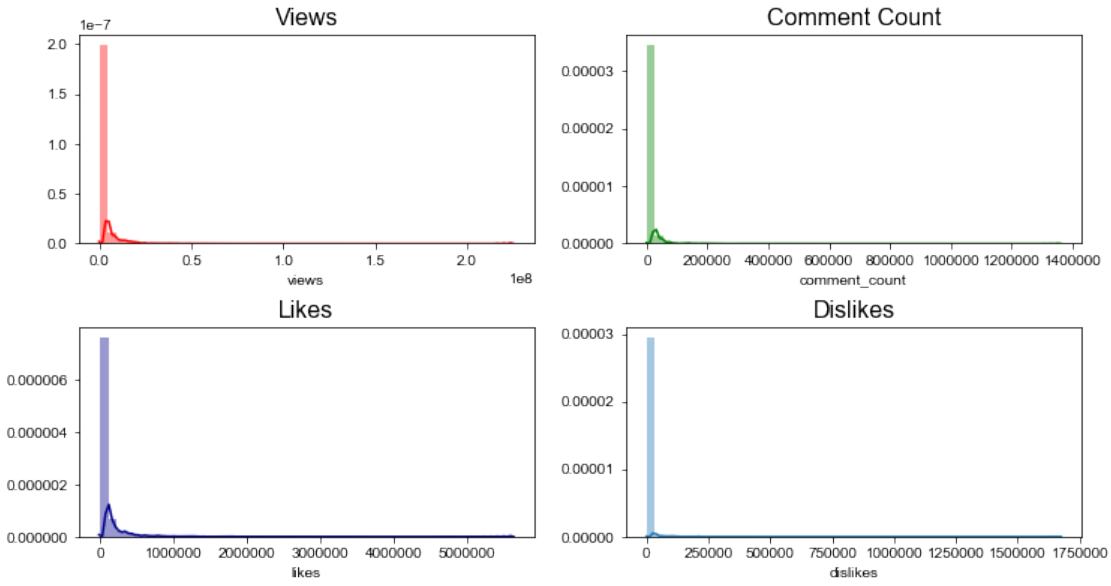
To get some information about the metrics, we first look into the descriptive statistics. Our metrics are the numerical variables in the dataset.

[9]:
```
# View the descriptive statiscs about the data
df.describe()
```

[9]:
```
              views         likes      dislikes  comment_count
count  4.094900e+04  4.094900e+04  4.094900e+04   4.094900e+04
mean   2.360785e+06  7.426670e+04  3.711401e+03   8.446804e+03
```

```
std     7.394114e+06  2.288853e+05  2.902971e+04   3.743049e+04
min     5.490000e+02  0.000000e+00  0.000000e+00   0.000000e+00
25%     2.423290e+05  5.424000e+03  2.020000e+02   6.140000e+02
50%     6.818610e+05  1.809100e+04  6.310000e+02   1.856000e+03
75%     1.823157e+06  5.541700e+04  1.938000e+03   5.755000e+03
max     2.252119e+08  5.613827e+06  1.674420e+06   1.361580e+06
```

[10]:
```
plt.figure(figsize = (12,6))

plt.subplot(221)
views_plot = sns.distplot(df["views"], color="red")
views_plot.set_title("Views", fontsize=16)

plt.subplot(222)
comment_plot = sns.distplot(df["comment_count"], color="green")
comment_plot.set_title("Comment Count", fontsize=16)

plt.subplot(223)
likes_plot = sns.distplot(df["likes"], color="darkblue")
likes_plot.set_title("Likes", fontsize=16)

plt.subplot(224)
dislikes_plot = sns.distplot(df["dislikes"])
dislikes_plot.set_title("Dislikes", fontsize=16)

plt.subplots_adjust(wspace = 0.2, hspace = 0.4,top = 0.9)

sns.set()
sns.set_color_codes("dark")
plt.show()
```



From the descriptive statistics and graphs above, we can see that the data is heavily skewed and does not underlie normal distribution. To process with the data, we will have to normalize it. We will do it in the next step.

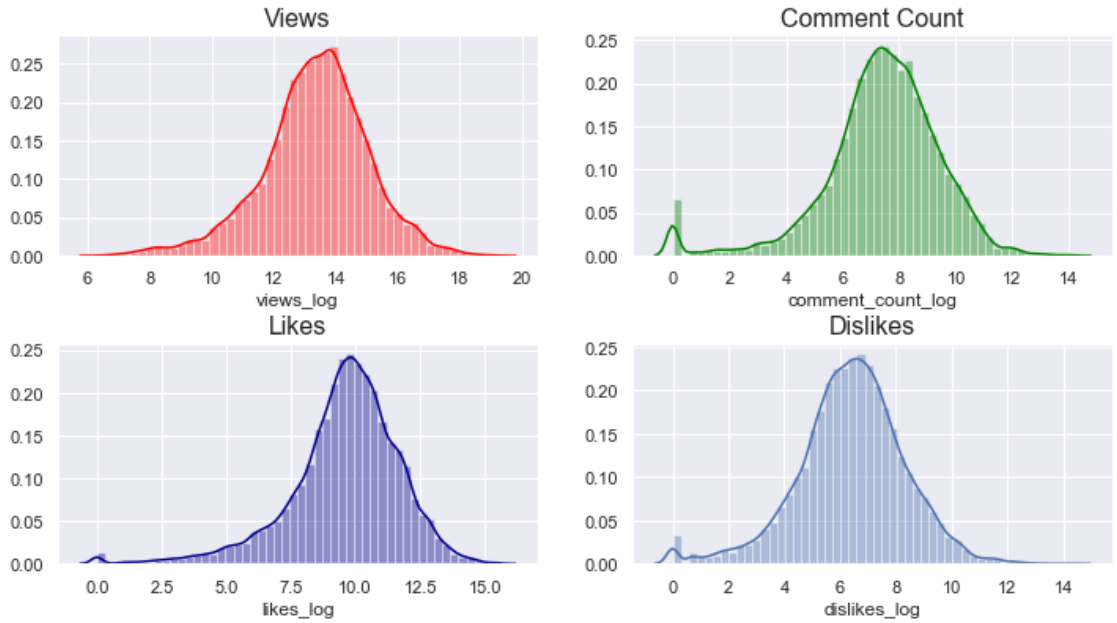
2.7 2.6 - Normalizing the numerical variables

To create a new measure, we should compare the distribution of the numerical variables in the dataset. Obviously we have some outliers, so taking the natural log of a value can reduce the variation caused by extreme values.

[11]:
```
df["views_log"] = np.log(df["views"] + 1)
df["likes_log"] = np.log(df["likes"] + 1)
df["dislikes_log"] = np.log(df["dislikes"] + 1)
df["comment_count_log"] = np.log(df["comment_count"] + 1)
```

After we have applied the transformations, we check the distributions graphs

[12]:
```
plt.figure(figsize = (12,6))

plt.subplot(221)
views_plot = sns.distplot(df["views_log"], color="red")
views_plot.set_title("Views", fontsize=16)

plt.subplot(222)
comment_plot = sns.distplot(df["comment_count_log"], color="green")
comment_plot.set_title("Comment Count", fontsize=16)

plt.subplot(223)
likes_plot = sns.distplot(df["likes_log"], color="darkblue")
likes_plot.set_title("Likes", fontsize=16)

plt.subplot(224)
dislikes_plot = sns.distplot(df["dislikes_log"])
dislikes_plot.set_title("Dislikes", fontsize=16)

plt.subplots_adjust(wspace = 0.2, hspace = 0.4,top = 0.9)

sns.set()
sns.set_color_codes("dark")
plt.show()
```



---

3 3 - Methodology

Since our dataset contains four metrics likes, dislikes, views and comment count, but we want one metric, which decides about the trending strength, we have to calculate it from existing metrics. Thus, creating a derived metric is an approach to create a new metric which we will need for the analysis but which do not exist in the data source. In the second step, we will design a machine learning pipeline for a YouTube video title processing. The pipeline has to be used chain multiple transformations into one. This is useful as there is some steps required in processing the text data, such as cleaning, normalization, tokenization, removing stop words and lemmatization. Then we transform the data and put it into the predictor. To choose the right estimator we used the machine learning map from Scikit-Learn.

3.1 3.1 Creating a new metric "viral index"

After looking into data, we try to determine what could control the trending probability (we call it "viral index"). After some research, we define the viral index as a logarithm of the sum of views, likes and comment counts minus 2 factor dislikes, devided by the highest rank. We weighted dislikes more, because dislikes are havier than the number of positive likes. According to the Verge "Reports have suggested that a video with a high number of dislikes — that outweighs the number of positive likes — is less likely to be recommended, and could therefore hurt the creator". Likes and views seem to have a strong correlation with each other, so we might renounce to weight the likes stronger than views. Similar applies to the comments. After we sum up the variables, we apply the logarithm to handle the outliers.

[13]:
```
highest_rank = (df["views"] + (df["likes"] - 2* df["dislikes"]) + ␣
↪df["comment_count"]).sort_values(ascending=False).iloc[0]
```

[14]:
```
df["viral_index"] = np.log((df["views"] + (df["likes"] - 2* df["dislikes"]) + ␣
↪df["comment_count"]) / highest_rank)
```

After we created the new metric called viral index, we look into some statistics:

[15]:
```
df["viral_index"].head(10)
```

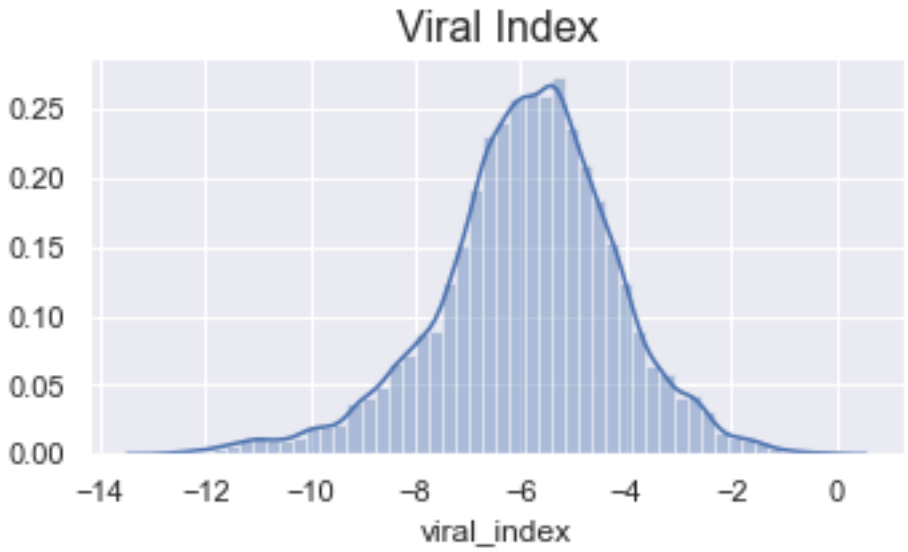
[15]:
```
0   -5.641799
1   -4.515543
2   -4.233900
3   -6.476388
4   -4.631216
5   -7.483559
6   -4.688605
7   -5.608831
8   -5.625050
9   -6.752442
Name: viral_index, dtype: float64
```

[16]:
```
df["viral_index"].describe()
```

[16]:
```
count    40949.000000
mean        -5.881301
std          1.711087
min        -12.920595
25%         -6.815322
50%         -5.789953
75%         -4.802419
max          0.000000
Name: viral_index, dtype: float64
```

We plot the distribution of the new metric viral index, to check if it is normalized.

[17]:
```
plt.figure(figsize = (12,6))
plt.subplot(221)
g1 = sns.distplot(df["viral_index"])
g1.set_title("Viral Index", fontsize=16)
sns.set()
sns.set_color_codes("dark")
```



[18]:
```
df.sort_values(by="viral_index", ascending=False).head(10)["title"]
```

[18]:
```
38547    Childish Gambino - This Is America (Official V...
38345    Childish Gambino - This Is America (Official V...
38146    Childish Gambino - This Is America (Official V...
37935    Childish Gambino - This Is America (Official V...
37730    Childish Gambino - This Is America (Official V...
37531    Childish Gambino - This Is America (Official V...
37333    Childish Gambino - This Is America (Official V...
37123    Childish Gambino - This Is America (Official V...
36913    Childish Gambino - This Is America (Official V...
36710    Childish Gambino - This Is America (Official V...
Name: title, dtype: object
```

3.2 3.2 Design a machine learning model based on natural language processing

Here we create our model using the new metric viral index. We take the raw title of a youtube video, clean it, normalize it, and convert it into a form that is suitable for the word embeddings. Word embeddings are vector representations of a particular word. After that we put the word embeddings to a the machine learning model, fit its parameters to training data, and use an optimization procedure. At the end, we save the model to a pickle format to use it to make predictions about unseen data later in our app.

[19]:
```
# define X and Y data sets
X = df["title"]
Y = df["viral_index"]
```

[20]:
```
# create the text processing function
def tokenize(text):
    text = re.sub(r"[^a-zA-Z0-9]", " ", str(text))
```

```
    tokens = word_tokenize(text)
    lemmatizer = WordNetLemmatizer()
    default_stopwords = set(stopwords.words("english"))
    tokens = [w for w in tokens if w not in default_stopwords]
    clean_tokens = []
    for tok in tokens:
        clean_tok = lemmatizer.lemmatize(tok).lower().strip()
        clean_tokens.append(clean_tok)

    return clean_tokens
```

It is important, to look into the pipeline we have chosen. It consists of a CountVectorizer, TfidTransformer, and SVR (Support Vector Regression) as a predictor. CountVectorizer provides a simple way to both tokenize the YouTube titles and build a vocabulary of known words, but also to encode new YouTube titles using that vocabulary. TfidTransformer performs the TF-IDF transformation from a provided output matrix of CountVectorizer. The SVR is based on the computation of a linear regression function in a high dimensional feature space where the input data are mapped via a nonlinear function.

[21]:
```
#create a pipeline for the machine learning model
pipeline =  Pipeline([
    ('vect', CountVectorizer(tokenizer = tokenize)),
    ('tfidf', TfidfTransformer()),
    ('predictor', SVR(kernel ="rbf", gamma='scale', C=1.0, epsilon=0.2))
])
```

[22]:
```
#divide the data into testing and training set
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state = 42)
pipeline.fit(X_train, Y_train)
```

[22]:
```
Pipeline(memory=None,
         steps=[('vect',
                 CountVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>, encoding='utf-8',
                                 input='content', lowercase=True, max_df=1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 1), preprocessor=None,
                                 stop_words=None, strip_accents=None,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=<function tokenize at 0x1a283c1d90>,
                                 vocabulary=None)),
                ('tfidf',
                 TfidfTransformer(norm='l2', smooth_idf=True,
                                  sublinear_tf=False, use_idf=True)),
                ('predictor',
                 SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.2,
                     gamma='scale', kernel='rbf', max_iter=-1, shrinking=True,
                     tol=0.001, verbose=False))],
```

```
            verbose=False)
```

[23]:
```
# check an Youtube title, if the model predicts any value.
msg = ["YouTube Rewind: The Shape of 2017 | #YouTubeRe..."]
pipeline.predict(msg)
```

[23]:
```
array([-4.27965595])
```

3.3 3.3 - Optimizing and evaluating the model

To tune the model parameter, there is a handy method in Scikit Learn called GridSearchCV. The GridSearchCV is used to train our model with multiple combinations of training hyper parameters and finds the best combination of parameters which optimizes the evaluation metric. It creates an exhaustive set of hyperparameter combinations and train model on each combination. In the second step we evaluate our model using model evaluation function and r2 square. R2 square is a metric which devided the total variance explained by model to the total variance of a model. A low value would show a low level of correlation, meaning a model that is not valid in most cases. The high value shows a high level of correlation, meaning the model is valid.

[24]:
```
# create parameters for an optimization procedure
parameters = {
        'vect__ngram_range': ((1, 1), (1, 2)),
        'tfidf__use_idf': [True, False],
        'tfidf__norm': ['l1', 'l2']
        }
# optimize model
model = GridSearchCV(pipeline, param_grid=parameters, cv=2, verbose=1, n_jobs=4)
model.fit(X_train, Y_train)
```

```
Fitting 2 folds for each of 8 candidates, totalling 16 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  16 out of  16 | elapsed:  5.2min finished
```

[24]:
```
GridSearchCV(cv=2, error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('vect',
                                        CountVectorizer(analyzer='word',
                                                        binary=False,
                                                        decode_error='strict',
                                                        dtype=<class
 'numpy.int64'>,
                                                        encoding='utf-8',
                                                        input='content',
                                                        lowercase=True,
                                                        max_df=1.0,
                                                        max_features=None,
                                                        min_df=1,
                                                        ngram_range=(1, 1),
```

```
                                                    preprocessor=None,
                                                    stop_words=None,
                                                    strip_accents=None,
                                                    tok...
                                            SVR(C=1.0, cache_size=200, coef0=0.0,
                                                degree=3, epsilon=0.2,
                                                gamma='scale', kernel='rbf',
                                                max_iter=-1, shrinking=True,
                                                tol=0.001, verbose=False))],
                                    verbose=False),
             iid='warn', n_jobs=4,
             param_grid={'tfidf__norm': ['l1', 'l2'],
                         'tfidf__use_idf': [True, False],
                         'vect__ngram_range': ((1, 1), (1, 2))},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=1)
```

[25]:
```
#use the model to predict the values
y_pred = model.predict(X_test)
y_pred.item(1)
```

[25]:
```
-5.924000607446279
```

[26]:
```
# check the model metrics
print(model.score(X_test,Y_test))
print(r2_score(Y_test,y_pred))
```

```
0.8407424098419297
0.8407424098419297
```

[27]:
```
# save the model to disk
joblib.dump(model, 'model.pkl')
```

[27]:
```
['model.pkl']
```

---

4 4 - Results

4.1 4.1 - Prepering the results for the app and saving into database

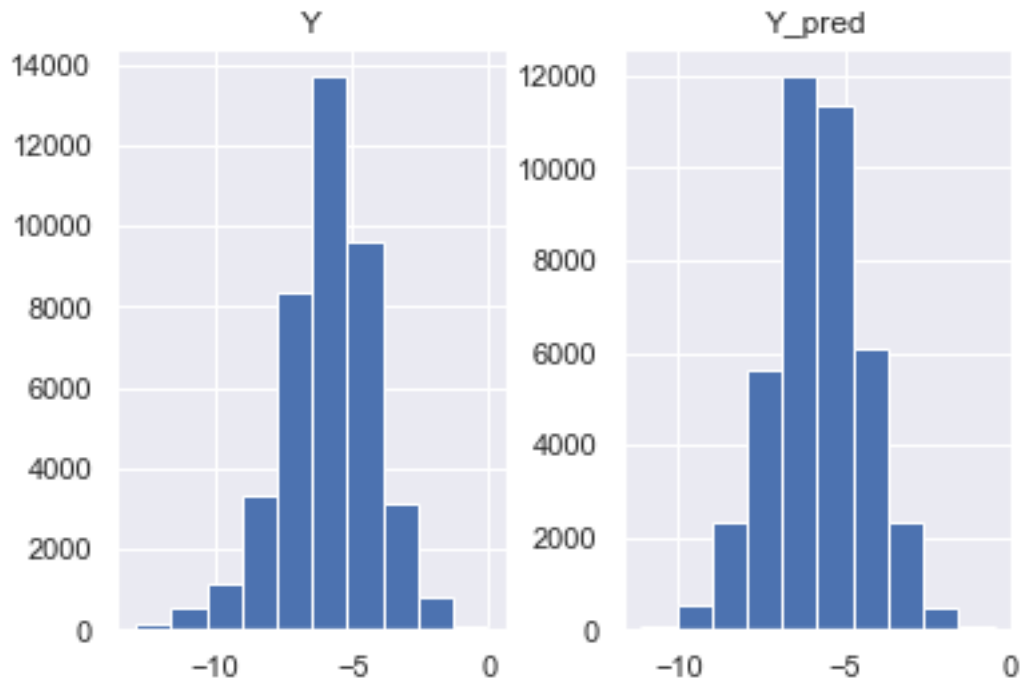
To prepare the results for the app, we devide the results into four equal-sized groups using quantiles. With the four groups we create a four steps scale. To each quantile, we assign a name "bad < ok < promising < superior". Later we use the 4 quantiles to make predictions about unseen data in our app.

[28]:
```
result = pd.DataFrame()
result["X"] = X
result["Y"] = Y
result["Y_pred"] = pipeline.predict(X)
```

[29]:
```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///USvideos.db')
result.to_sql("trends_data", engine, index=False)
```

[30]:
```
result.hist(bins=10)
```

[30]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a2a324dd8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2c13d7f0>]],
      dtype=object)
```



[31]:
```
def create_labels():
    labels = ["bad", "ok", "promising", "superior"]
    return labels

def create_bins(df):
    limit0 = df["Y_pred"].quantile([.0]).iloc[0]
    limit25 = df["Y_pred"].quantile([.25]).iloc[0]
    limit50 = df["Y_pred"].quantile([.5]).iloc[0]
    limit75 = df["Y_pred"].quantile([.75]).iloc[0]
    bins = [limit0, limit25, limit50, limit75, np.inf]
    return bins
```

[32]:
```
predicted_value = model.predict(["My new YouTube Video about Fallout"])
labels = create_labels()
bins = create_bins(result)
# create a label based on quantile value
```

```
pd.cut(predicted_value, bins, labels=labels)[0]
```

[32]: `'promising'`

---

5 5 - Conclusion

5.1 5.1 - Reflection

We implemented a Flask app, that rates a new video title on a trending scale. To do it, we have used the trending data from YouTube and transformed in into a new metric. We also used the natural language processing algorithms, to analyze and classify new YouTube video title on a trending scale. The difficulty of the project was to find the best machine learning pipeline, also to implement the app, so that the app can rate the new video title based on the model built with this notebook.

5.2 5.2 - Improvement

Unfortunately YouTube trending algorithm is not public. Hence we could only make assumptions to what extent user interactions are part of it. Our app also rates only the title of the video. The data also contains the thumbnail of the trending video. Possible further analyses could include the image classification, if the data also could derive any information about the trends. Futher we could also include the new title a user has added into the app to our machine learning model.