W251 Final Project Report

A rating system for videos published on the social media sites Twitter, Facebook and Youtube user social media activity.

April 26, 2016

W251 Scaling Up! Really Big Data
Megan Jasek, Andrea Soto, Alejandro Rojas, Charles Kekeh

Table of Contents

Introduction

The goal of our final project was to identify popular videos that engage users across social media platforms. To do this, we built an application that ingests video metadata from Twitter, Facebook and Youtube, and analyzes the content on two fronts: a batch mode and a stream mode.

The application works with the different platforms, connects to their API, parses the JSON response from each platform and stores the data in a MongoDB data store, while also adhering to the terms of use and rate limits of each API. With the data in a database, we can display the 10 most popular videos on each site and how the popularity of the video has evolved. Finally, we did some off-line analytics which include some exploratory data analysis, sentiment analysis, and prediction model.

To make our corpus of data, we continuously query the platforms for a list of predefined topics and filter each post to extract the video information relevant for our analysis.

Background

Back in 2011, well-known investor Mark Suster predicted that the future of the Internet will be video. His reasoning was very straight forward: in the US people spend more than 5 hours a day watching TV and less than 1 hour reading[1]. Extrapolating that proportion to online behavior provided a certain guidance that video will come to dominate the Internet.

In the last 5 years, video growth has been phenomenal. A fact that has not been missed by major social networks. Facebook, Google and Twitter are all devoting significant resources to grow their video presence betting that capturing user's attention requires increasing the amount of video content at their disposal. Today, Facebook and Youtube claim to have billions of video views a day.[2]

Content creators have also been scrambling to churn out video content that strikes a note with their audiences. New media outlets like BuzzFeed and Upworthy are increasingly focusing on developing video content that becomes viral on social networks. With the rise of smartphones, consumers are also adding their set of video content to the mix. All this growth in video content publication and consumption is creating a need that our application wants to attend.

---

[1] *See some discussion at Mark Suster's blogentry:* *http://www.bothsidesofthetable.com/2013/09/17/how-online-video-companies-can-increase-margin-and-build-better-businesses/*

[2] *See some discussion about video views on Facebook and Youtube:* *http://www.forbes.com/sites/edmundingham/2015/04/28/4-billion-vs-7-billion-can-facebook-overtake-youtube-as-no-1-for-video-views-and-advertisers/2/*

In the space, there are startups like Datamnr[3] and Crowded Tangle[4] that are serving media companies with products that identify social media posts that are getting attention but none of them are exclusively focusing on video.

Value Proposition

Questions to answer

Online media publishers need to know what type of video content is getting attention across major social media networks like Facebook, Twitter and Youtube, Instagram, Vine and Snapchat.

Online media publishers also need to be able to know how their video content did when it was published on major social media networks against video content published by others.
Our application wants to help media publishers answer: what video content to publish? when to publish? and where?

Product description

Our application creates a platform to show rankings of videos posted on social networks, report ratings on how video content performed once it was published and predict whether a video will become popular or not.

Our application continuously ingest metadata from videos published on three major social networks: Twitter, Youtube and Facebook. Publishers can see how popularity metrics evolve over time on each site. It also uses data collected to create analytical and machine learning tools to help predict the virality of recently published videos.

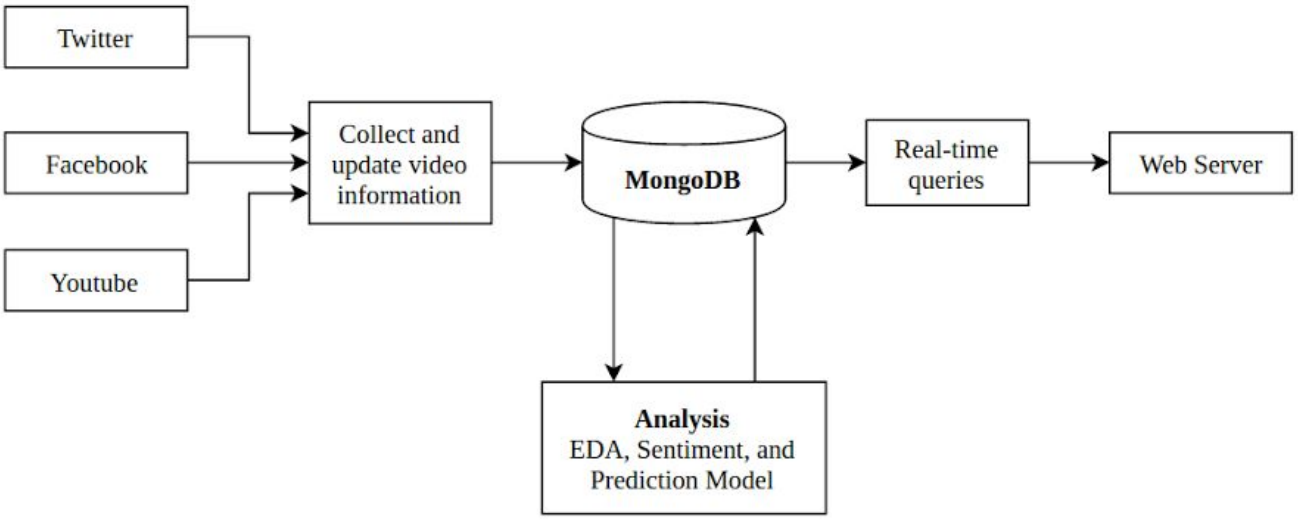
---

[3] *See article about Datamnr here:*
http://www.cjr.org/analysis/the_new_importance_of_social_listening_tools.php

[4] *See article about Crowded Tangle here:*
http://www.fastcompany.com/3040951/the-secret-tool-that-upworthy-buzzfeed-and-everyone-else-is-using-to-win-facebook

Architecture

The overall architecture of the project is shown in the figure below.



The data was collected from three sources: Twitter, Facebook, and Youtube. The independent ingestion processes connected to the API's of each source and gathered metadata on videos.

The data was parsed and stored in a single node, MongoDB database. MongoDB provided the following advantages:

- It is easy to install, implement and develop. With the pymongo library, we could easily connect and query the database.
- It favors consistency over availability. Since one goal of the application was to provide a ranking of popular videos, it was important to show the same results to users, even if the results were not completely up-to-date.
- It is a document store with a dynamic schema. This was important because the video information provided by the APIs was not uniform. For example, if a publisher did not add a description to the video, then the 'description' field would not be included in the API's response. This made it hard to define a fixed schema and validate documents before writing them to a traditional relational database. So having a database with a dynamic schema was a desirable feature. With a dynamic schema we could also continue ingesting data even if the APIs changed.
- It provides easy javascript querying.
- Can scale horizontally.

A separate collection was created for each source to make it easier to manage the data being collected, with secondary indexes created to speed-up frequent queries.

The web server connects to MongoDB to show the most popular videos for each source in real-time. The analysis was done off-line, and includes some exploratory data analysis and a prediction model.

Ingestion Processes

The ingestion of data had to be customized for each source because each API was different. Twitter and Facebook are not video sites, so the ingestion of video metadata was not as straightforward as with Youtube, where we could directly query for recently posted videos.

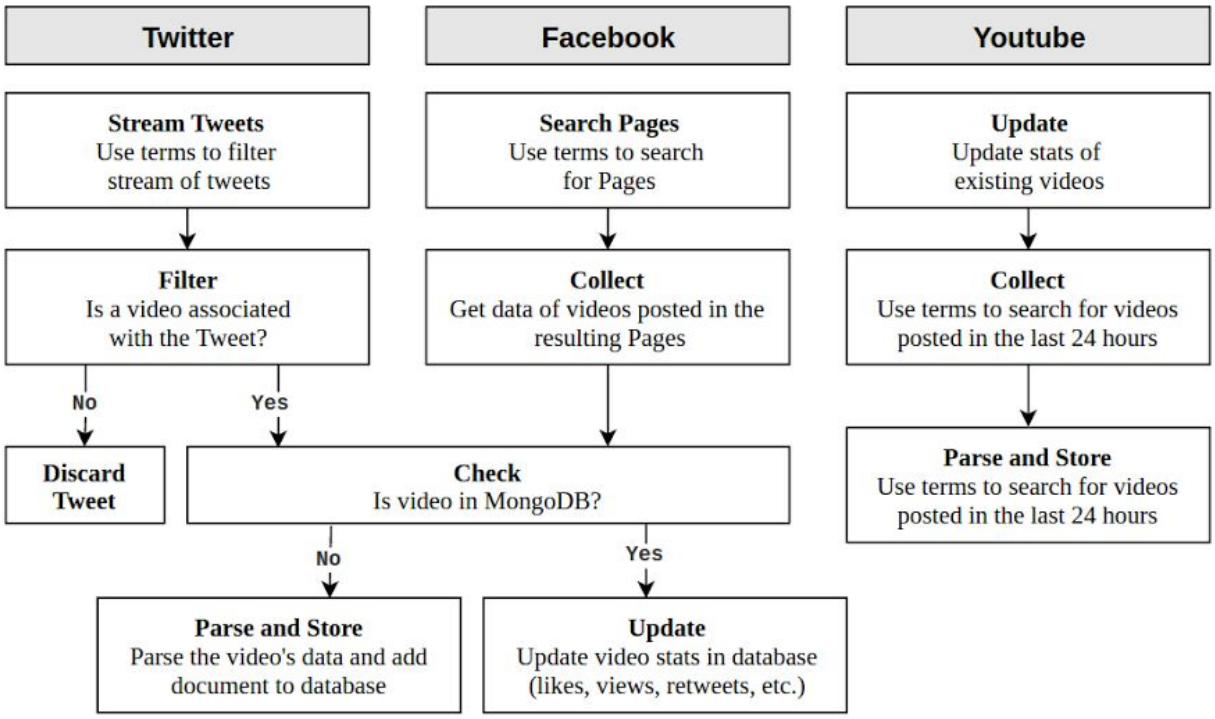
For Twitter, we used the Streaming API to get a stream of tweets. We filtered the tweets that had video content associated to it and stored their metadata in MongoDB. For Facebook, we used the Graph API to search for public pages and get all the videos posted by each page. For Youtube, we use the Data API to search for videos posted in the last 24 hours.

We used a set of 78 terms related to entertainment, music, sports, and food to query all the sources. We did this for several reasons:

- The selected topics have a lot of video content published and many popular videos are in one of these categories[5].
- We wanted to increase the chances of finding tweets with video.
- We needed a search term to query the Facebook API.
- We wanted to have similar content across the three sources.

[5] Based on a Google search of popular videos

An overview of the data collection processes is shown in the diagram below.



The ingestion processes for Twitter and Facebook were continuous, whereas the Youtube process was scheduled to start every 24 hours by a Cron scheduler and would stop when finished.

In general, the ingestion of data was relatively slow. While twitter supports streaming, only about 1% of tweets have video, so most tweets were discarded. On the other hand, Facebook and Youtube do not have streaming services and the rate limits of their APIs restricted our capacity to ingest data. The rate limits also meant that we had to find a balance between gathering new data and updating the data of existing videos. The update process was important because it gave us the trend of popularity counts, including number of retweets, likes, comments, and views.

Twitter did not have an update process; instead, it relied on re-encountering existing videos to update the counts. For facebook, two parallel process were run; the first process would collect data on new videos and the second process would make the updates by passing through the same videos as the first process. The second process was run a week after the first process. For YouTube, updates took place at the start of the process, so the counts were collected every 24 hours.

Data

We collected data from Twitter, Facebook, and Youtube for a period of 4 weeks, starting on March 25, 2016. The data that we collect from the 3 sources is summarized in the table below.

Source	Videos	Size (GB)
Twitter	691,250	6.8
Facebook	985,000	12.9
Youtube	11,802	0.9
Total	**1,688,052**	**20.6**

Analysis and Prediction

One of our main goals was to predict whether a newly posted video would become popular or not. Since our data was not labeled, we created a binary labeled based on the following popularity counts:

- Retweets for Twitter videos
- Likes for Facebook videos
- Views for Youtube videos

If a video's popularity measure was above a threshold it would be labelled it as popular (1) otherwise it would be labelled as not popular (0).

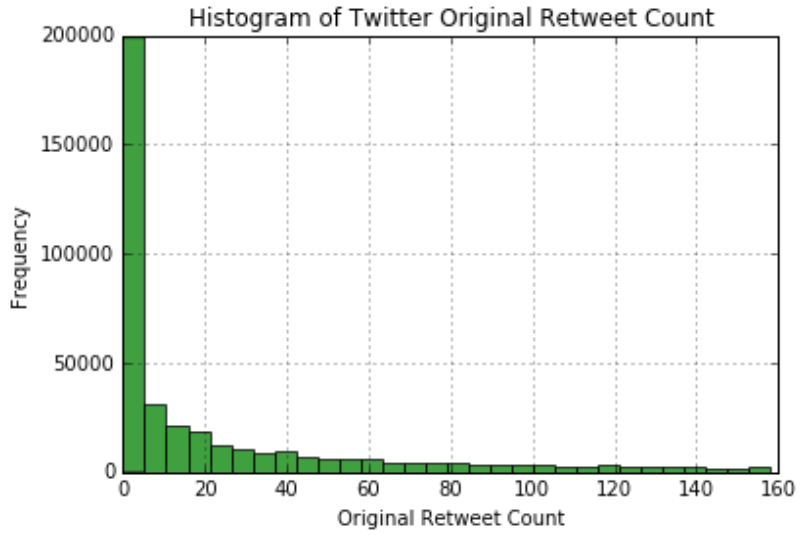
Exploratory Data Analysis

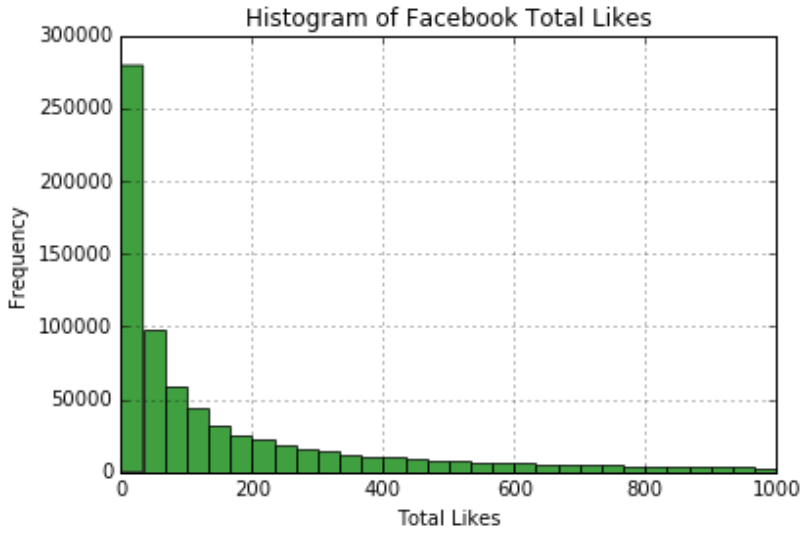
We analyzed the data gathered in the document DB to determine the distribution of popularity counts for the measures of popularity selected across each data source. Determining the appropriate level to decide whether a video is popular provided us the ability to automatically label our data set so we can train a classifier on it to predict whether a new video in the system will become popular.

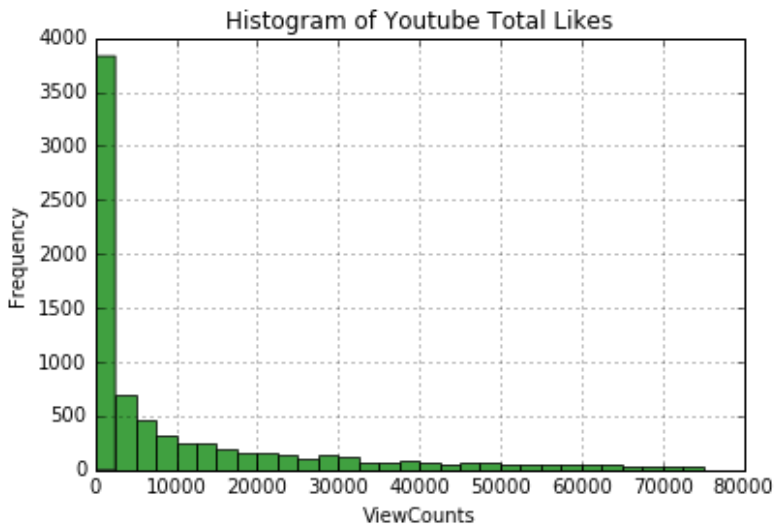
The histograms of the data across the three data sources we collected are displayed below. For each data source, the 75th percentile was chosen as the cutoff point to determine the popularity of a video from that source.

The respective quartiles for each distribution are as follows:

Data Source	1st quartile	2nd quartile	3rd quartile
Twitter	1	17	140
Facebook	21	100	495
Youtube	407	6,801	50,790







Predicting Popularity

The project uses the data from all 3 data sources to make a prediction about how popular a video will be.

Model Features and Output

The model chosen for predicting popularity is logistic regression because we are predicting a categorical (binary) variable with exactly 2 values: 1 for popular and 0 for not popular.

- Features. The features selected for the model are as follows:
    - Video_length_sec - the length of the video in seconds
    - Popularity_count - the measure of popularity defined for each source as follows:
        - twitter = retweet_count, facebook = total_likes, youtube = view_count
    - Other_count - another count defined for each source as follows:
        - twitter = favorite_count, facebook = total_comments, youtube = favorite_count
    - Growth_rate - the Popularity_count divided by Hours_Alive
        - Hours_Alive is defined as the time of the last update of a video minus the time it was first created.
    - Sentiment - sentiment score of a text string from each video as predicted by vaderSentiment. The text string varies by source as follows.
        - twitter = tweet text, facebook = video description, youtube = video description
        - vaderSentiment is a python library that returns a sentiment score based on a string of text.
    - Source - either “twitter”, “facebook” or “youtube”
- Model output (formula). The resulting formula that was created for the model is as follows:

- Popularity = 0.0 - 155.6 * Video_length_sec + 16.1 * Popularity_count + 13.3 * Other_count - 0.48 * Growth_rate - 0.43 * Sentiment - 7.37 * Source

Model Implementation, Creation and Prediction

The model was implemented using the MLlib module of Spark and the Python programming language. MLlib provides machine learning classes for use with Spark RDDs and other Spark data structures. The main features used in MLlib were LabeledPoint, LogisticRegressionWithSGD, LogisticRegressionModel.

- LabeledPoint - data type that allows easy modeling of an outcome label (in this case 1 or 0) and a list of features (those listed above).
- LogisticRegressionWithSGD - class that implements the logistic regression model and stores it.
- LogisticRegressionModel - class that enables manipulation of a created model which includes loading a model from a file.

The model was created with the following process:

1. Separate JSON data files from twitter, facebook and youtube were loaded into a Spark RDDs. The JSON files were created using data from the MongoDB from March 25, 2016 to April 4, 2016.
2. Data that did not contain all of the required features was filtered out and not used in the model training. This could sometimes happen if one of the source APIs did not give complete data for a video.
3. A custom method called create_labeled_points_<source> was called to create the MLlib LabeledPoint data structures for each source based on the data for each video.
4. Data was split into a training set and a test set. First, data from each source was split into a training set and a test set with 80% of the data in the training set and 20% in the test set. Then each of the source-specific sets were combined. This ensured that there was the same proportion of data from each source in the training and test sets.
5. The LogisticRegressionWithSGD was called to create the model. This process usually took about 90 minutes using 1.1 million videos. Then LogisticRegressionModel was called to save the model to a file.
6. The model was then evaluated on both the training set and the test set and a training error was recorded. The training error recorded for both the training and test set was about 11%. This means that the model was getting 11% of its predictions incorrect.

After the model was created a new set of unseen data was pulled from the MongoDB to make popularity predictions on videos. The following process was used:

1. The pymongo Python library was used to load data from MongoDB. The data used for the predictions were videos collected from April 5, 2016 and after.
2. Predictions of popularity were made on the data by loading the stored model using LogisticRegressionModel.

3. Predictions were written back to the MongoDB using pymongo. The predictions were stored in a new field called ‘prediction_logistic_reg’.

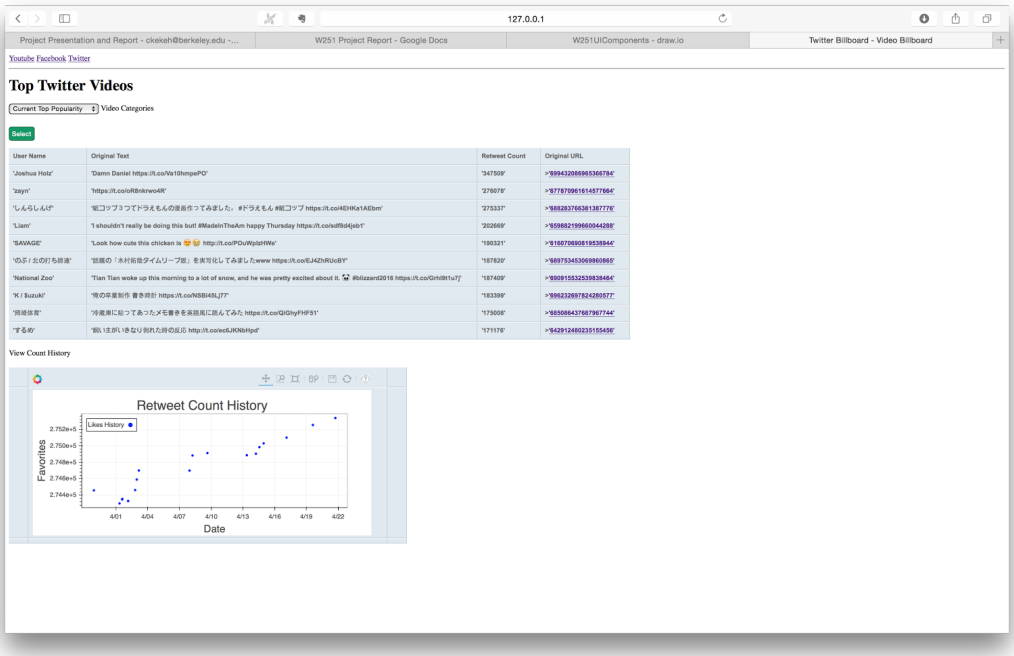
Prediction Model Code Example - MLlib from Spark and Python

The following is a generalized coding excerpt from the method that creates the model from the training data. The code has been modified for readability. It is intended to give the reader a sense of the MLlib code required to create the prediction model.

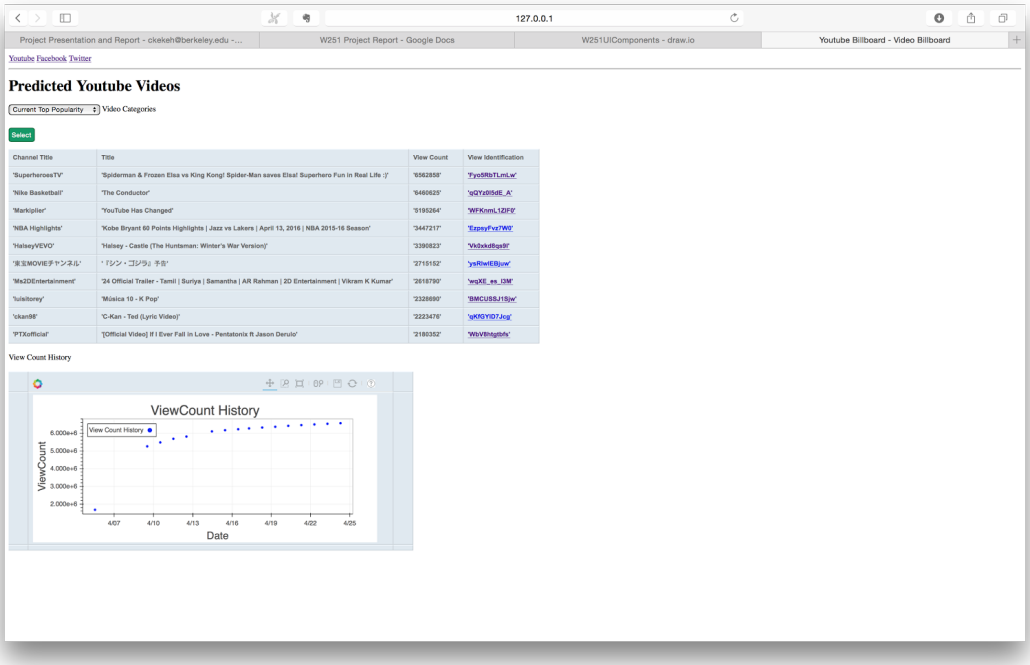
```
# Create Spark Context
sc = SparkContext(appName="SparkCreateModel")
# LOAD data from JSON files into Spark RDDs by source
twitter_data = load_data_from_file(sc, "file:///root/mongoData/twitter.json")
youtube_data = load_data_from_file(sc, "file:///root/mongoData/youtube.json")
facebook_data = load_data_from_file(sc, file:///root/mongoData/facebook.json")
# CREATE  MLLib LabeledPoints (LP =  LabeledPoint(DependentVar, [FeaturesList]))
twitter_LP = twitter_data.map(create_labeled_points_twitter)
youtube_LP = youtube_data.map(create_labeled_points_youtube)
facebook_LP = facebook_data.map(create_labeled_points_facebook)
# SPLIT DATA into training (80%) and test(20%) sets
train_twitter, test_twitter = twitter_LP.randomSplit([0.8, 0.2], seed=0)
train_youtube, test_youtube = youtube_LP.randomSplit([0.8, 0.2], seed=0)
train_facebook, test_facebook = facebook_LP.randomSplit([0.8, 0.2], seed=0)
# COMBINE all 3 datasets with the RDD.union command
train_LP = train_twitter.union(train_facebook).union(train_youtube)
test_LP = test_twitter.union(test_facebook).union(test_youtube)
# BUILD MODEL -  logistic regression
model_log = LogisticRegressionWithSGD.train(train_LP)
if store == True:
       model_log.save(sc, file_path)
# EVALUATE MODEL on test data
preds_test_log = test_LP.map(lambda p: (p.label, model_log.predict(p.features)))
```

Front-End UI

The following screen captures provide details of our project UI for two scenarios selected. The first capture is from the display of most popular twitter videos. The second, similar in the display to the first one reports the most popular Youtube videos that were predicted for popularity.

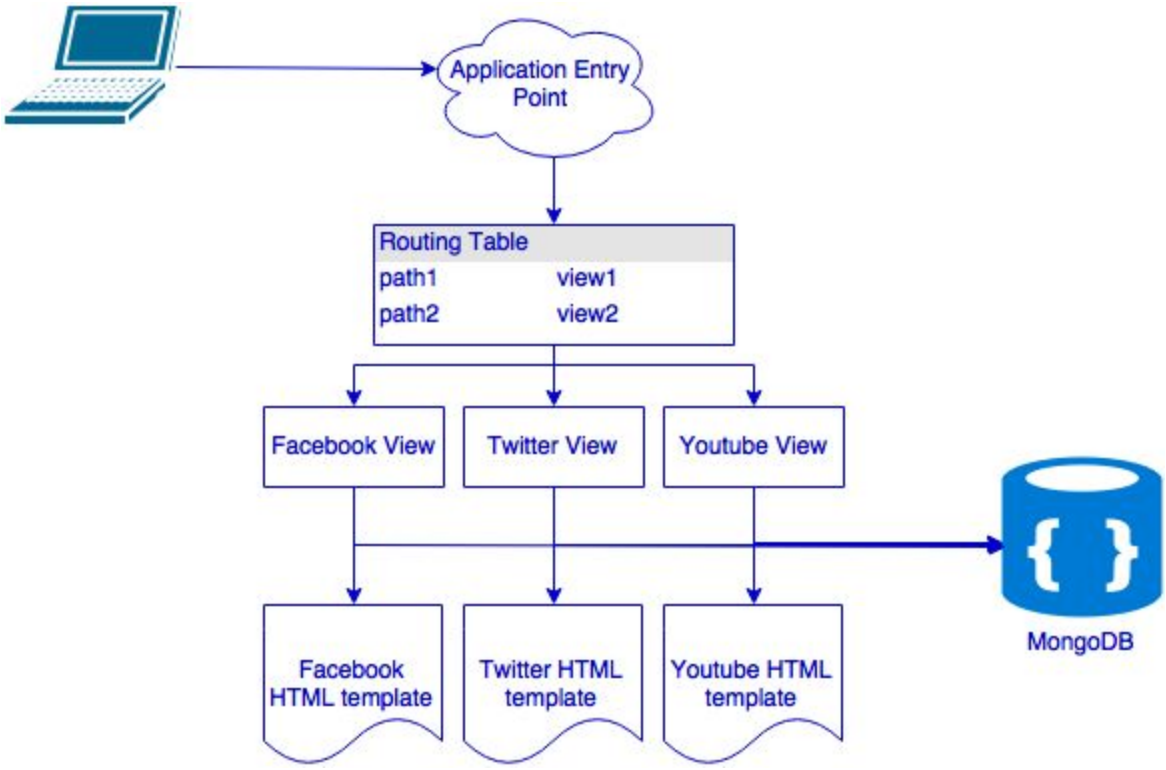


Top Twitter videos by popularity with details of popularity history on a top video



Youtube videos predicted for popularity ordered by view count

The diagram below identifies the main components of the user interface Web application. Those are the endpoint, the routing table, the views, the templates those views render, and the connection to the document database.



The routing table is a mapping of the application URLs to views that handle the requests for those URLs. The views in turn gather data that is then rendered using pre-existing HTML templates. The data gathered is collected from the document DB, given the parameters of the HTTP request coming through the routing table to the views. The separation between views and templates provides reasonable separation of responsibilities between business logic and presentation.

Challenges, Limitations, and Future Work

Challenges

- Variety: early on we realized that we needed to get familiarized with the data that could be collected from each source. Because the sources were so different, we had to customize the collection and cleaning pipeline for each one. This also had an impact on all of our processes downstream. We needed to reconcile the different fields available in each dataset and find ways to work with the three sources seemingly.

- API Rate Limits: We set out to collect a large amount of video data, but soon realized we were rate limited by the API and that we would need to ingest data for more than 2 weeks to get a sufficiently large dataset for the project.

Limitations

Our current implementation has the following limitations:

- The data is limited to the 76 terms used to search and filter. We are not collecting data on videos outside this scope.
- Our single node Mongo database worked well for the implementations, but could become a bottleneck as more data is added.
- Currently, we cannot identify the same video across sources.
- Our prediction model is categorical (popular/not popular) and uses features that are highly correlated to the variable used to label the data.

Future Work

- Identify the same video across the three sources to have a better indicator of popularity of videos.
- Use characteristics of the video and topic tags as features in the prediction model.

Product Roadmap

We envision our product evolving on three fronts:

1. **Tracking Tool:** a tool targeting publishers and advertisers that display key metrics that they want to track to make key decisions in terms of content to fund and audiences to target.
2. **Image Classification Tool:** a tool complementing current video's metadata with a deeper analysis of the images that the video contains. This will include not only the ability to identify the same video across different networks and different users but also a transformation of the images into features that can serve as input for better predictive tools.
3. **Predictive Tool:** a more robust tool that can precisely identify windows of opportunities of when and where to post specific videos for specific audiences. Potentially a predictive tool that can help content producers identify "viral" features to include when producing content.

Project Structure

The code for the project is located at https://github.com/abessou/w251-FinalProject

- code
    - Mongo_scripts

        - Different MongoDB scripts used during the course of the project and not required to replicate this project
    - Sinks
        - The implementation of different ‘sinks’ or stores for the data ingested. In the end, the MongoDBDataIngestSink was selected as the only sink used for this project.
    - Sources
        - The implementation of the different ingestion ‘sources’ in use in this project. Of those, the ones selected for the final implementation of this project are the FacebookDataIngestSource, the TwitterDataIngestSource and the YoutubeDataIngestSource.
    - Spark
        - The implementation of the classifier training and prediction Spark code.
    - Web_client
        - The implementation of the web application for the project UI
- configFiles
    - Configuration files used by each ingestion source to drive the common ingestion process that hosts every ingestion source. The common ingestion host is hosted in ‘DataIngest.py’ and should be called with a configuration file as a parameter.
- data
    - Sample data that’s not required to replicate the project.
- images
    - Images and documentation that’s not required to replicate the project.
- notebooks
    - Notebooks that were used for exploratory work on the document store.
- setup
    - Bash scripts that install the python library dependencies for each component.