

# Forecasting Video Analytics

Sami Abu-El-Haija, Ooyala Inc (haija@stanford.edu; sami@ooyala.com)

## 1. Introduction

Ooyala Inc provides video Publishers an end-to-end functionality for transcoding, storing, and delivering videos to a wide set of devices. Publishers can easily upload their video content and make it available on websites, smart phones, tablets, Facebook Applications, and Google TVs. In addition, publishers can access a wealth of Analytics data on how their video content is being watched.

Ooyala offers daily-aggregated Analytics. For example, the number of plays a video received on a given day. The aim of this research is to analyze historical video analytics, model the time-series of video plays, and predict the number of plays that a video is going to receive over the next two days.

We begin this paper by describing the source of our data, and our proposal for reducing the inherent noise in the data set (Sections 2 and 3). Afterwards, we describe the various models we explored, and reason our choices (Section 4). Then, we compare the accuracy achieved by the models (Section 5). Lastly, we suggest future work based on our insights and conclude our findings (Sections 6 and 7).

## 2. Data sets

The goal of this research is to predict analytics at a *daily granularity* (Analytics granularity currently offered to our publishers). However, we construct our models on data more granular than a day. Our intuition tells us that sub-day patterns will allow us to better-predict analytics for day granularities. Therefore we ran MapReduce jobs to accumulate data at **3-hour granularity**. Hence, one day consists of 8 buckets.

## 3. Gaussian Smoothing

With 3-hour buckets, the number of plays for a video across time is noisy.



Figure 1: Video A – real plays ( $\hat{P}$ )

To reduce noise, we apply a Gaussian filter:

$$P_t = \sum_s W(t-s) * \hat{P}_s$$

Where  $\hat{P}_s$  is the observed (real) number of plays at time  $s$ ,  $P_t$  is the smoothed number of plays, and,

$$W(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{\sigma^2}\right)$$

For computational efficiency, we set  $W(x)$  to approximate a discrete Gaussian distribution with mean 0, that hard-drops to 0 for  $|x| > 2$ , and sums up to 1. In particular:

$$W(0) = 0.435; \quad W(-1) = W(1) = 0.241; \\ W(-2) = W(2) = 0.083; \quad W = 0 \text{ elsewhere.}$$

Our models performed significantly better when trained with smoothed data

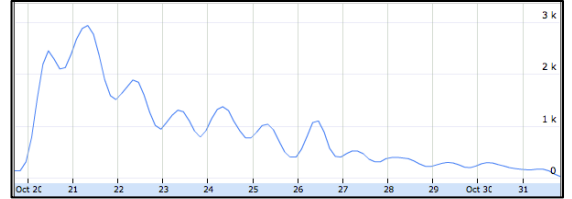


Figure 2: Video A – smoothed plays ( $P$ )

## 4. Models

It is observed from the dataset that different videos whiteness different viewership patterns. For example, videos published by News publishers receive a spike of interest in their early lifetime, but the number of plays quickly drops over time. On the other hand, long-content videos observe more of a uniform pattern of plays over time. In addition, it is observed that most videos whiteness time cycles. Some videos are more popular at night; others are more popular during the day.

In the remainder of this section, we describe the different models we explored. We split the models into two categories, Per-Video Models and Per-Publisher Models.

#### 4.1 Per-Video Models

To overcome trend differences between videos, these models look at each video in isolation, fitting the model parameters per video, where we:

1. Leave-out the last two days of the data set
2. Train the model on the rest of the data
3. Compute the error between the predicted and the actual number of plays for the last two days.

##### 4.1.1 Linear Model

Here, we model  $P_t$  as a linear combination of the historical plays ( $P_{t-1}, P_{t-2}, \dots$ )

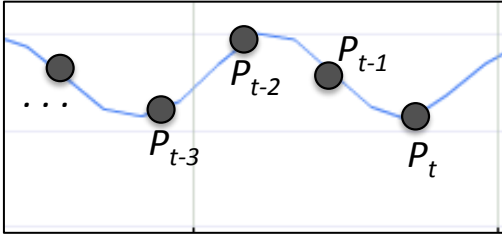


Figure 3: Time-series

$$P_t = \theta_1^0 * P_{t-1} + \theta_2^0 * P_{t-2} + \dots + \theta_{T_0}^0 * P_{t-T_0}$$

Where  $T_0$  is a configurable constant, and  $\theta^0$ 's are the model parameters, written compactly:

$$P_t = \sum_{i=1}^{T_0} \theta_i^0 * P_{t-i}$$

To fit the model parameter  $\vec{\theta^0}$  for a video, we generate training examples by “traversing” over the historical timeline of the video. As we want to relate plays ( $P_t$ ) to  $T_0$  historical values, we start this traversal at  $t = T_0$ , and for each increment of  $t$ , we extract one training example with features  $P_{t-1}$  through  $P_{t-T_0}$  and value  $P_t$ . We leave out the last two days of the dataset for prediction.

Then, we compute  $\vec{\theta^0}$  from the learning examples using closed-form linear regression.

Finally, the above formula relates  $P_t$ , with the set of *directly* preceding  $P$ 's ( $P_{t-1}$  through  $P_{t-T_0}$ ). In a predictive setting, the previous  $P$ 's will be given, and if we have  $\theta^0$ , we can estimate  $P_t$ . However, in addition to estimating  $P_t$ , we would like to estimate  $P_{t+1}, P_{t+2}, \dots, P_{t+15}$ , such that we can estimate the number of plays a video will receive over the **next 2 days**. Therefore, we construct 16 models, one model to predict each future point. In particular, we estimate 16  $\theta$ 's satisfying:

$$P_{t+j} = \sum_{i=1}^{T_j} \theta_i^j * P_{t-i}; \quad j \in [0, 15]$$

##### 4.1.2 Autoregressive Moving-Average Model

Here, we explore the classic Autoregressive Moving Average (ARMA) time-series model (Box, Jenkins, Reinsel, 1994) on our dataset.

The ARMA( $p, q$ ) has two components. The Autoregressive (AR) part, parameterized by  $p$ , is similar to the *Linear Model* described in the previous subsection, which models future values in terms of previous values. The Moving-Average (MA) part, parameterized by  $q$ , models latent noise in the time-series. We define ARMA( $p, q$ ) for our data set as:

$$P_t = \sum_{i=1}^p \theta_i * P_{t-i} + \sum_{j=1}^q \phi_j * \varepsilon_{t-j}$$

Where  $\varepsilon$ 's are white noise. Here, we used the `arima()` function in **R** to estimate the parameters of the model [3].

*Note: we dropped the intercept term from the ARMA model by setting `include.mean=FALSE`. Its presence degraded our prediction quality.*

#### 4.2 Per-Publisher Models

In these models, we train on each publisher in isolation. Our motivation is to capture general trends on a publisher's videos. If we can accurately model the general trends of a publisher, we could predict plays on recently uploaded videos using the publisher's general trends.

### 4.2.1 k-means clustering

Our driving intuition behind this model is to find *averages of shapes of curves* that well-describe a video publisher.

We would like to encode a segment of a time-series as a vector, such that two vectors have a low Euclidean distance if the shapes of their respective segments were *similar*. Moreover, we want the vector representation to be scale-invariant. Some videos are inherently more popular than others. Nonetheless, two News videos, for instance, are likely to take a similar path (e.g, the number of plays during daytime is roughly double than during nighttime).

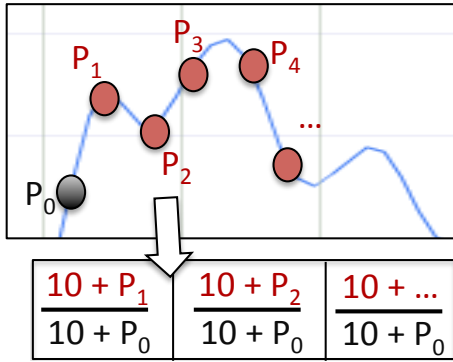


Figure 4: Depicting vectorization

In the vectorization process, depicted above, we compute the vector for the segment  $[P_1, P_2, \dots]$ , where each component  $i$  of the vector is the ratio of  $(10 + P_i) / (10 + P_0)$ , where  $P_0$  is the number of plays preceding the segment. The choice of 10, *the normalization constant*, is to remove noise coming from *dead videos*. For instance, if a video received 1 play in a 3-hour window, followed by 9 plays in the next window, followed by 1 play, could greatly skew the general publisher's curve shapes if we indicated a 900% increase followed by a 900% decrease. We feel that 10 would absorb-away vibrations when the number of plays is less than 10, and the constant will have little effect on the ratio of  $(P_i / P_0)$  when the number of plays is averaging well over one hundred.

In this model, we fixed the width of the segment to be six days.

To extracting learning examples, given a publisher, we traverse its videos' timeseries to

produce thousands of *shape vectors*. Then, we run the k-means clustering algorithm on the vectors to find  $k$  vectors (centroids of clusters) that well-describe the publisher. Below are two cluster means computed for two publishers (here, we construct a time-series from shape vectors by reversing the computation, setting  $P_0 = 50$ )

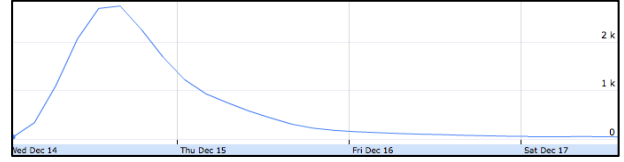


Figure 5: Centroid for a News publisher. Many of its other centroids looked similar with varying the peak positions and varying width of curve.

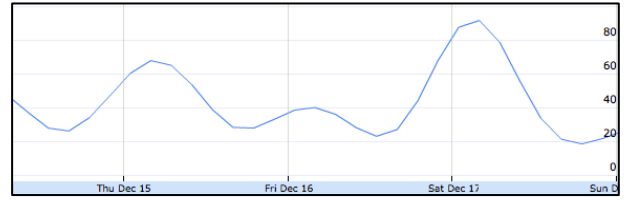


Figure 6: Centroid for a long-content publisher, which shows more-or-less a consistent, cyclical pattern.

In a predictive setting, we have computed the  $k$  clusters for a publisher, and we are given a video curve with two segments: a known, *filled* segment, and an *unfilled*, to-be-predicted segment. We fix the width of the filled segment 4 days, and the unfilled segment to 2 days, such that the total width of the curve is 6 days, matching the clusters widths. Next, we vectorize the known segment (just like before). Then, we choose the cluster that minimizes the Euclidean distance between the first 3 days of the cluster vector and the known vector. Finally, we fill the unfilled segments according to the cluster vector.

This model can be summarized by the pseudo-code (applied separately to each publisher):

```

Train(timeseries_training_set)
1. vectors := vectorize(timeseries_training_set)
2. clusters := k_means(vectors, k)

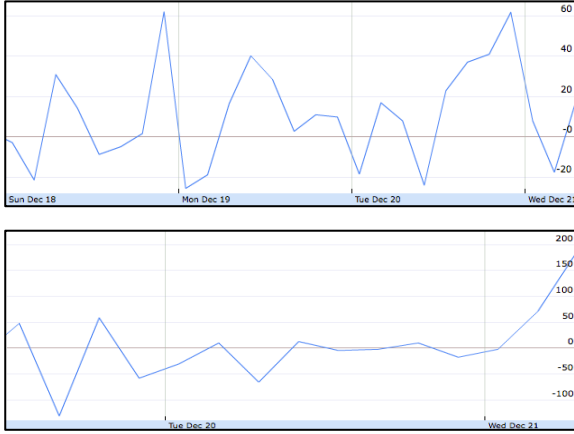
PredictPlays(video)
1. s := vectorize(timeseries(video, last 4 days))
2. cluster := argmin_c ||4 days1c - s||
3. ts_2days := ForecastUsingShapeVector(s, cluster)

```

### 4.2.2 Principal Component Analysis

The intuition behind applying Principal Component Analysis (PCA) comes from visual inspection that features of a shape vector are highly correlated. For example, if the number of plays is increasing between 7:00 AM and 11:00 AM, then after going through a daily cycle, the number of plays likely to be also increasing the next day, between 7:00 AM and 11:00 AM.

We used PCA to reduce the dimension of the shape vector from 48 (6 days) to 16. Then, we ran k-means on the reduced dimensions to find  $k$  averages. Finally, mapped the  $k$  clusters back to 48 features for the purpose of visualization and inspection, and they looked like:



Figures 7 & 8: Centroids after applying PCA and mapping back to original dimensions.

It is apparent from the centroids figures above that our application of PCA poorly modeled our data. Most centroids looked very similar to the ones above (noisy, with lots of negative numbers). After this visual inspection, we stopped our exploration of PCA to model our data set.

## 5. Results

To measure the accuracy of the models, we,

1. Quantitatively measure the prediction error over two (unseen) days that follow the training set
2. Qualitatively, visually inspect the shape of the predicted curve, and see the extent it follows the true (actual) curve.

Let  $P_t^*$  denote the predicted plays at time  $t$ . In addition, for notational convenience, let  $D_1$  denote the number of plays over the first unseen day, and  $D_2$  denote the number of plays over the first and the second unseen days:

$$D_1 = \sum_{t \in \text{day1}} P_t; \quad D_2 = \sum_{t \in \text{day1} \cup \text{day2}} P_t$$

Similarly, let  $D_1^*$  and  $D_2^*$  be the same summations over predicted plays ( $P_t^*$ ).

We define the prediction errors as:

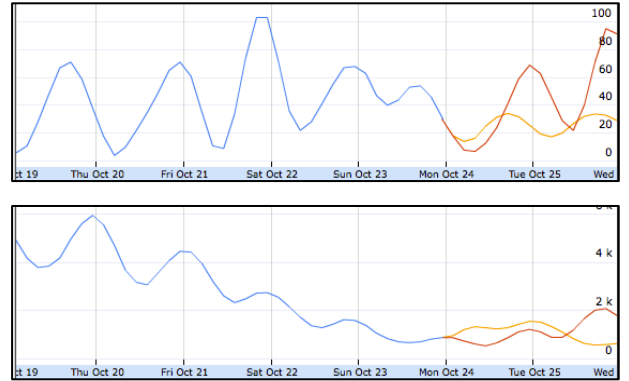
$$\text{error}(1 \text{ day}) = \frac{|D_1 - D_1^*|}{\max(D_1, D_1^*)}$$

$$\text{error}(2 \text{ days}) = \frac{|D_2 - D_2^*|}{\max(D_2, D_2^*)}$$

Moreover, we will use a consistent legend in plotting time-series. We will plot the portion seen by our algorithms in blue, the predicted line in orange, and the true (actual) line in red.

### 5.1 Linear Model

We set  $T_j$ 's = 2 days, yielding predictions:



Figures 9 & 10: two typical prediction curves.

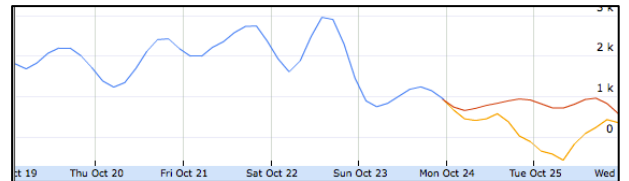


Figure 11: Rare (but visible) case where the model predicts negative numbers for a time period.

Over a set of randomly chosen videos, the linear model produced **29.9%** and **31.7%** errors on one and two day predictions, respectively.

## 5.2 ARMA Model

The ARMA model had the best performance on our data set. Some prediction graphs:

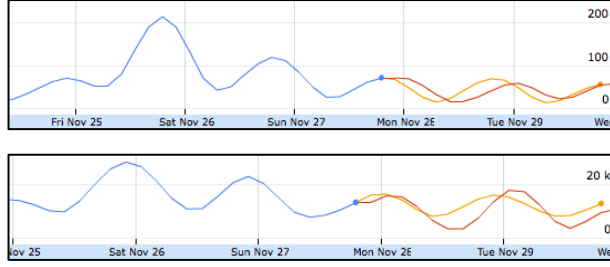


Figure 12: Prediction curves for ARMA(16, 4)

Varying  $p$  and  $q$ , average errors were:

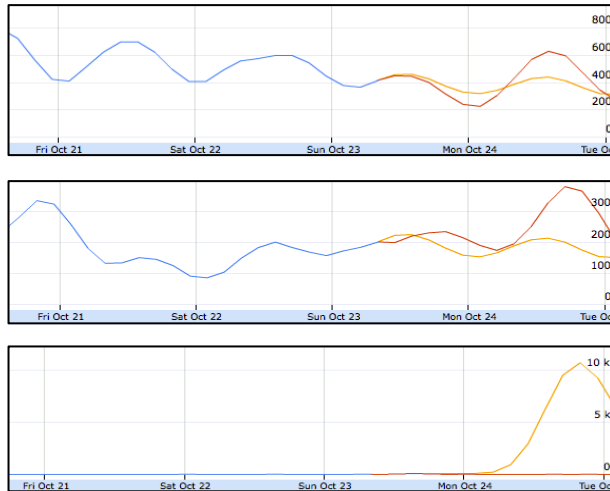
| $p$       | $q$      | error(1 day)  | error (2 days) |
|-----------|----------|---------------|----------------|
| 14        | 4        | 17.72%        | 18.25%         |
| <b>16</b> | <b>4</b> | <b>13.88%</b> | <b>15.73%</b>  |
| 16        | 8        | 16.05%        | 25.04%         |
| 17        | 4        | 15.71%        | 18.01%         |
| 17        | 8        | 20.60%        | 24.14%         |
| 23        | 3        | 18.77%        | 20.21%         |

Table 1: Error averaging over a number of randomly chosen videos

It is worth noting that some videos achieve better predictions with smaller  $p$  and  $q$ , while others achieve better predictions with larger  $p$  and  $q$ . In practice, there are algorithms that can be used to identify a good choice of  $p$  and  $q$  (such as, Box-Jenkins method [1]).

## 5.3 k-means clustering

For  $k=100$ , the prediction curves look like:



Figures 13, 14 and 15: The first two are common cases; predicted and actual curves have similar directions. The last is a rare case, where the known (blue) portion matches the cluster but cluster spikes on unseen section

Although in most cases, the predicted curve and the true curve have similar shapes, the accuracy of this model is much worse than the Per-Video models. We ran k-means twice, for  $k=50$  and  $k=100$ . Then, for a randomly selected set of curve shapes, we computed the average error when predicting for 1 day and 2 days:

| $k$        | error(1 day)  | error (2 days) |
|------------|---------------|----------------|
| 50         | 61.93%        | 65.72%         |
| <b>100</b> | <b>58.42%</b> | <b>61.84%</b>  |

## 6. Future Work

- Invest more in Per-Publisher models. Try Mixture of Gaussians model instead of k-means clustering.
- Model time-of-day and day-of-week
- Look at different sources of data, such as the countries that plays are coming from, or the viewer IDs that are watching the modeled videos.

## 7. Conclusion

Our tests have shown that our Per-Video models perform better than our Per-Publisher predictions. However, we strongly feel that it is possible to extract relationships between videos within the same publisher. Nonetheless, the described k-means and PCA approaches were unable to capture such relationships.

## 8. References

- [1] George Box, Gwilym M. Jenkins, and Gregory C. Reinsel (1994), *Time Series Analysis: Forecasting and Control*, third edition. Prentice-Hall
- [2] James D. Hamilton (1994), *Time Series Analysis*, Princeton University Press
- [3] Ripley, B. D. (2002) Time series in R 1.5.0. *R News*, 2/1, 2–7. [http://www.r-project.org/doc/Rnews/Rnews\\_2002-1.pdf](http://www.r-project.org/doc/Rnews/Rnews_2002-1.pdf)