Consumer Behavior in the Online Classroom:

Using Video Analytics and Machine Learning to Understand the Consumption of Video Courseware

Mi Zhou	Pedro Ferreira	Michael D. Smith	George H. Chen
UBC Sauder School of Business	CMU Heinz College	CMU Heinz College	CMU Heinz College
mi.zhou@sauder.ubc.ca	pedrof@cmu.edu	mds@cmu.edu	georgechen@cmu.edu

Web Appendix[1]

APPLICATION TO A DIFFERENT CONTEXT – CRASH COURSE

To alleviate potential concerns about the specific context of Masterclass.com, we apply our proposed video feature framework to a different context — Crash Course — an educational YouTube channel with over 12 million subscribers.[2] We have obtained a dataset from Crash Course consisting of 1,127 videos for 39 courses. Different from Masterclass.com, Crash Course material focuses on educational disciplines, such as statistics, computer science, physics, chemistry, biology, engineering, and ecology. Each course includes a different number of chapters and each chapter includes one video lecture. We also obtained a time-coded subtitle file for each Crash Course video. In addition to the course videos, we acquired a unique dataset from a third-party company which captures the historical consumption records for these videos.

We first use the same methods used in the body of the paper to extract the features for each video in our data. We then use the extracted video features to predict a different outcome variable in this setting — the popularity of the video one year after its posting, as extracted from the

[1] These materials have been supplied by the authors to aid in the understanding of their paper. The AMA is sharing these materials at the request of the authors.
[2] We thank an anonymous referee for this suggestion.

historical records. Since these videos were uploaded to YouTube at different times, we also include the uploaded time as a control variable in our analysis. The summary statistics of our data are reported in Table W1.

Table W1 Summary Statistics of Crash Course Videos

Variable	Mean	SD	Min	Max
Consumers' Viewing Behavior:				
Y_popular	0.499	0.500	0.000	1.000
Basic Video Properties:				
Chapter	19.966	13.371	1.000	51.000
Video Length	11.049	2.223	1.320	17.270
Average Scene Length	0.180	0.074	0.059	0.595
Speaking Rate	198.317	57.792	32.605	532.965
Sentiment	0.072	0.073	-0.240	0.341
Instructors' Emotions and Physical Characteristics:				
Anger	0.015	0.031	0.000	0.252
Contempt	0.011	0.013	0.000	0.095
Disgust	0.007	0.020	0.000	0.185
Fear	0.003	0.005	0.000	0.082
Happiness	0.176	0.141	0.000	0.780
Neutral	0.639	0.158	0.147	0.980
Sadness	0.039	0.055	0.000	0.303
Surprise	0.110	0.076	0.000	0.512
Age	35.553	5.727	21.474	50.038
Gender	0.724	0.447	0.000	1.000
Glasses	0.593	0.492	0.000	1.000
Facial Hair	0.160	0.158	0.000	0.588
Baldness	0.133	0.143	0.000	0.795
Hair Color	2.177	1.706	1.000	6.000
Makeup	0.350	0.393	0.000	1.000
Smile	0.176	0.141	0.000	0.780
Visual Aesthetic Features:				
Foreground Motion Area	0.347	0.090	0.147	0.625
Motion Magnitude	0.701	0.229	0.278	1.792
Motion Direction	3.122	0.081	2.493	3.444
Warm Hue Proportion	0.646	0.156	0.246	0.968
Saturation	0.381	0.095	0.138	0.666
Brightness	0.541	0.112	0.188	0.892
Contrast of Brightness	0.201	0.031	0.124	0.310
Clarity	0.987	0.030	0.492	1.000

We measure the popularity of the videos using a binary variable extracted from the historical records data: whether the video *v* in course *c* is popular among consumers ($Y\_popular_{cv}$). Specifically, for each video, if its total number of "likes" within one year after it was uploaded to YouTube is larger than the median, then we define $Y\_popular_{cv} = 1$; otherwise $Y\_popular_{cv} = 0$. We thus use $Y\_popular_{cv}$ as a proxy for $Y\_complete_{icv}$ in the MasterClass context. Note that there is still some difference between these two contexts. In the MasterClass setting, we have individual-level consumption records from a proprietary dataset, whereas in Crash Course we have only aggregate-level consumption records for each video.

Our empirical analysis follows the approach used for the MasterClass content. First, we use machine learning algorithms to investigate the extent to which our proposed video feature framework predicts the consumers' consumption of online course videos on Crash Course. We then use interpretability approaches in machine learning, including a standard feature permutation strategy (Breiman 2001, Molnar 2018) and Shapley values (Shapley 1953, Štrumbelj and Kononenko 2014, Lundberg and Lee 2017), to assess feature importance, which provides additional insights into the impact of different features on the consumers' viewing behavior.

In the first stage of our analysis, we use machine learning classifiers trained on our proposed video features to predict the behavior of consumers. As before, we use a gradient boosting machine (Friedman 2001) and use 80% of the data as training data, and the rest as test data (such that videos in the test data do not appear in the training data). Within the training data, we select hyperparameters using a 5-fold cross-validation prior to training on the complete training data (using the best hyperparameters found from cross-validation). When predicting whether a video is popular that received more likes from consumers ($Y\_popular_{cv} = 1$), our model achieves an average out-of-sample prediction accuracy of 85% (precision = 0.84, recall = 0.88, f1-score =

0.86, AUC = 0.92, ROC curve in Figure W1), which again suggests that our proposed video feature framework contains valid information that has predictive power of consumers' viewing behavior of online educational videos.

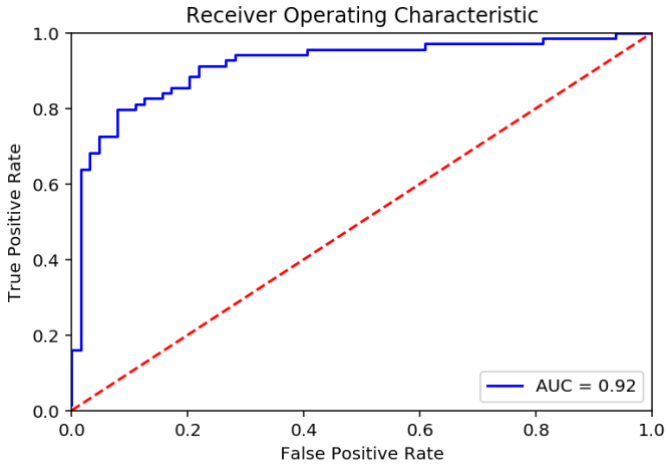

Figure W1 ROC curve and AUC

In the second stage of our analysis, we use the same machine learning interpretability methods to assess feature importance, which can provide additional insights into the impact of different video features on the consumers' viewing behavior. We first use a standard feature permutation strategy to assess category-level feature importance (Breiman 2001, Molnar 2018). Figures W2 presents results of permutation feature importance. We then compute Shapley values for the whole dataset to interpret individual-level feature importance (Lundberg and Lee 2017). Figures W3 summarizes the results of Shapley values.

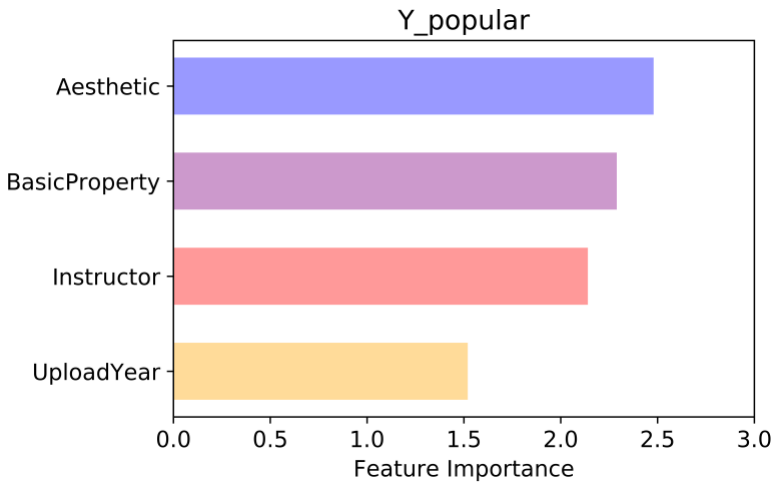

Figure W2 Category-Level Feature Importance

The results in Figure W2 confirm that video features are important factors in forecasting users' consumption behavior of online course videos. However, the aesthetic features are the most important in Crash Course videos, followed by basic video properties and instructors' characteristics. For example, the feature importance of aesthetic features is 2.5, meaning that permuting the values for aesthetic features more than double the prediction error in the model.

The results in Figure W3 show some interesting findings on the impact of the Crash Course video features in the Crash Course context. First, the uploaded time of the video is negatively related to the predicted outcome, suggesting that videos uploaded in recent years tend to be less popular (i.e., receiving fewer "likes") compared to videos uploaded in earlier years. For the video features, we find both similar and different patterns as compared to the MasterClass setting. For example, consistent with the findings in the MasterClass context, chapter has a negative impact such that later chapters in a course are correlated with lower likelihood of receiving "likes" from consumers. Most aesthetic features also show a positive impact in our predictive model. Specifically, videos with higher values of brightness, warmness, and clarity are predicted to be

more popular. Videos with lower values of brightness have a large negative impact on the predicted outcome. Speaking rate also has a negative impact, suggesting that when the instructor is speaking too fast in a video, the video is predicted to be less popular. Note that the average speaking rate in Crash Course videos (mean=198.32, sd=57.80) is higher than that of MasterClass videos (mean=144.05, sd=28.87). We also find some different impacts of video features in the Crash Course context. For example, sentiment has a negative impact on predicted outcome, which is contrary to the MasterClass setting. Such difference might be due to the different disciplines of the courses on these two platforms. Moreover, videos with very large motions have a negative impact, which might be due to the fact that on average Crash Course videos (mean=0.35, sd=0.09) have more motions than MasterClass videos (mean=0.20, sd=0.10), and very large motion values may cause a negative impact on consumers' "likes" of the video. To ensure the robustness of this analysis, we have also conducted additional analysis using "views" instead of "likes" as a measure for video popularity. Note that these two metrics are strongly correlated on YouTube. In our Crash Course data, the correlation between these two metrics is 0.93. Following the same procedures, our model achieves an average out-of-sample prediction accuracy of 83% when using "views" as a proxy for popularity. The feature importance analysis results are summarized in Figure W4, which show consistent patterns. Such consistency is likely due to the high correlation between "likes" and "views" on YouTube.

The differences between the specific predictors of consumption of Masterclass and Crash Course videos might be due to the differences in the types of content across the two platforms, the typical motivations of customers who use the two platforms, the revenue models (subscription vs. ad-supported), the production values, or the outcome variables available in the two settings

(individual-level viewing behavior for MasterClass videos vs. aggregate-level viewing behavior for Crash Course videos).

The main point is that our framework is able to accurately predict video consumption—and generate managerial insights—in both contexts, in spite of the differences in content, customers, revenue model, and outcome variables between the contexts. This suggests that our framework will generalize to, and be useful in, a variety of video consumption contexts that marketers may face online.

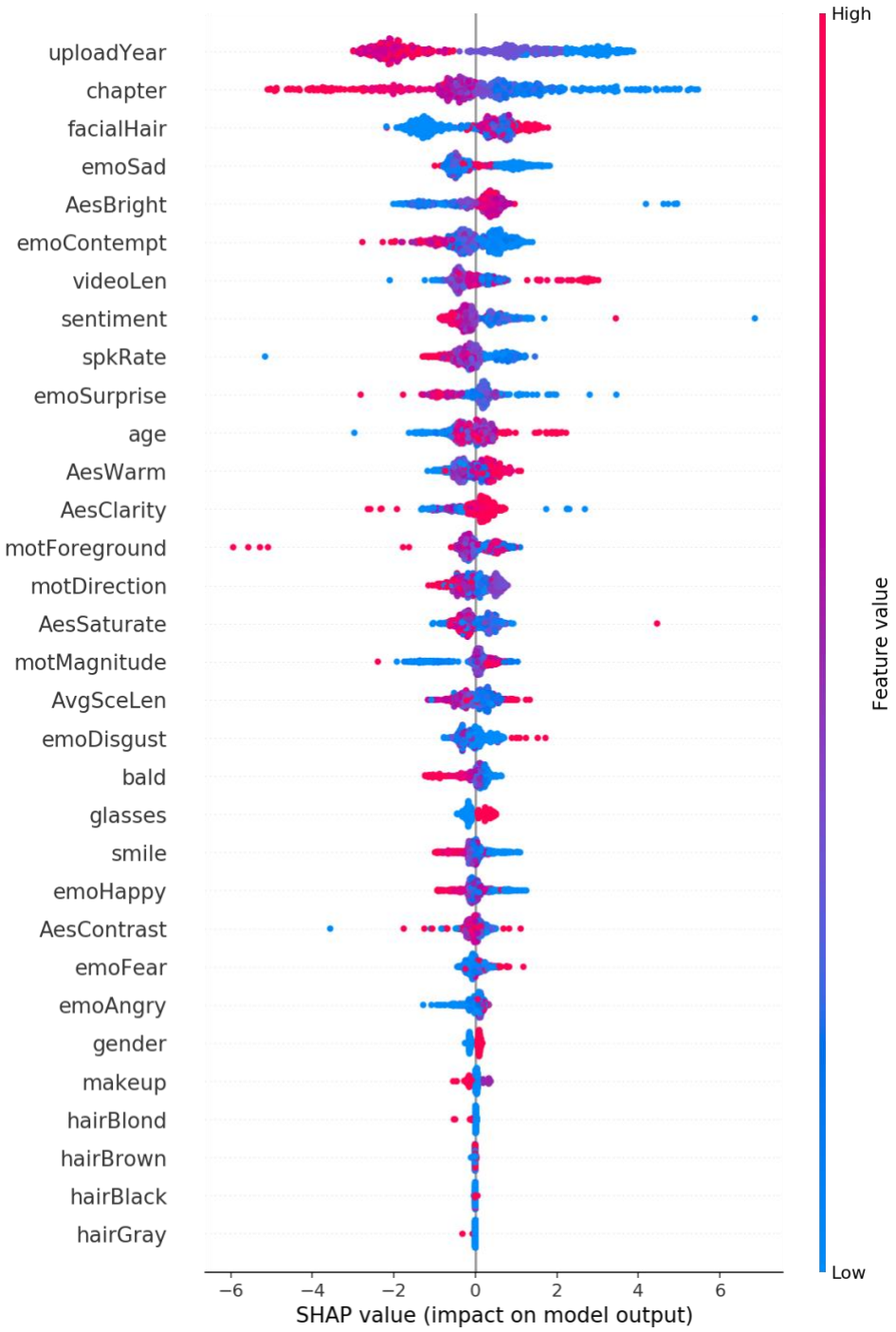

Figure W3 Individual-Level Feature Importance (using "likes" to measure popularity)

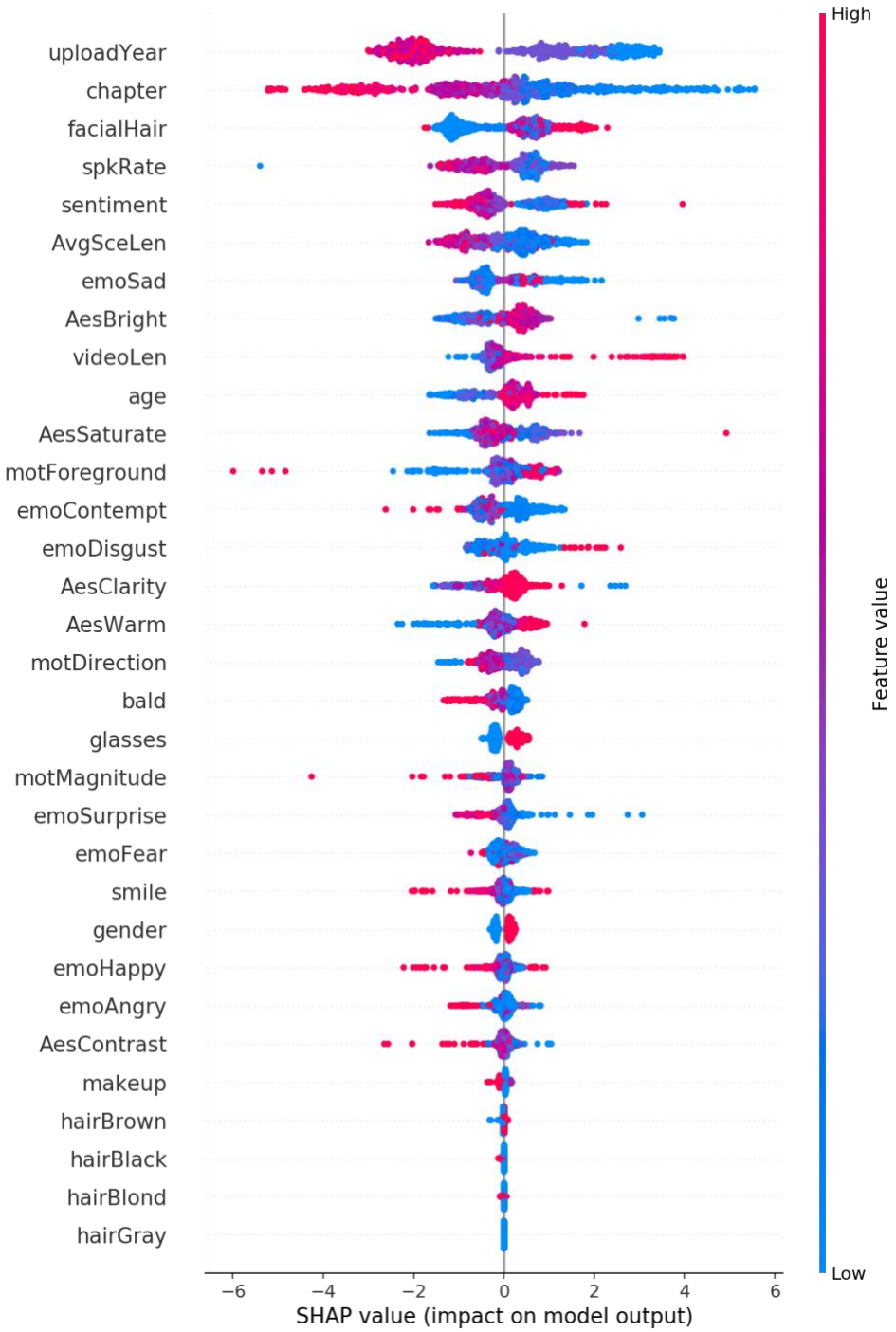

Figure W4 Individual-Level Feature Importance (using "views" to measure popularity)

PROPERTIES OF SHAPLEY VALUES

In the analysis we use a novel framework named SHAP (SHapley Additive exPlanations) (Lundberg and Lee 2017) to interpret individual-level feature importance. Specifically, SHAP uses a model agnostic representation of feature importance, where the importance of each feature is represented using Shapley values (Shapley 1953, Štrumbelj and Kononenko 2014, Lundberg and Lee 2017). Shapley values are borrowed from the cooperative game theory literature (Shapley 1953) and provide a theoretically justified method to allocate the output of a coalition among the coalition members (Štrumbelj and Kononenko 2014, Lundberg and Lee 2017). In our context, the coalition is a set of interpretable model input feature values (e.g., video features), and the output of the coalition is the value of the prediction made by the model when given the input feature values (e.g., consumers' viewing behavior).

The importance of each feature is defined as the change in the expected value of the model's output when the feature is observed versus unknown (Štrumbelj and Kononenko 2014, Lundberg and Lee 2017). Different feature values have different impact on the prediction. The Shapley values $\phi_j(f, x)$, explaining a prediction $f(x)$, are an allocation of credit among the various features in x (e.g., video features), and are the only such allocation that obeys two important properties: local accuracy and consistency (Young 1985, Lundberg and Lee 2017). Note that for a particular prediction, $\phi_j(f, x)$ is a single numerical value representing the impact of feature j on the prediction of the model f when given the input x. In our context, f is a gradient boosting model, and x is the set of input features including the video features and the consumer's past average video completion rate.

We provide a brief summary of the two desirable properties below following Lundberg et al. (2018). We refer to Lundberg and Lee (2017) for a full discussion and for relationships to

several other recent methods in complex model interpretability such as LIME (Local Interpretable Model-agnostic Explanations) (Ribeiro et al. 2016) and DeepLIFT (Deep Learning Important FeaTures) (Shrikumar et al. 2017). In the properties below, $f_x(S) = E[f(x) \mid x_S]$, where x_S is a subset of the input vector with only the features present in set S.

Local accuracy. The local accuracy property (also known as completeness or additivity) is given by

$$f(x) = \phi_0(f, x) + \sum_{j=1}^{M} \phi_j(f, x)$$

where $\phi_0(f, x) = E[f(x)]$ and M is the number of input features. The local accuracy assumption forces the Shapley values to correctly capture the difference between the expected model output and the output for the current prediction (Lundberg et al. 2018).

Consistency. For any two models f and f', if

$$f'_x(S \cup \{j\}) - f'_x(S) \geq f_x(S \cup \{j\}) - f_x(S)$$

for all $S \in Z/\{j\}$ where Z is the set of all M input features, then $\phi_j(f', x) \geq \phi_j(f, x)$. This states that if a feature is more important in one model than another, no matter what other features are also present, then the importance attributed to that feature should also be higher. Note that consistency is known as monotonicity in game theory literature.

Lundberg and Lee (2017) show that game theory results guaranteeing a unique solution apply to the entire class of additive feature attribution methods and propose SHAP values as a unified measure of feature importance that various methods approximate, including several popular methods such as LIME (Local Interpretable Model-agnostic Explanations) (Ribeiro et al. 2016) and DeepLIFT (Deep Learning Important FeaTures) (Shrikumar et al. 2017). They have also conducted a set of user studies which demonstrated that the SHAP method shows better

consistency with human intuition and more effectually discriminates among model output classes than several existing methods. We refer to Lundberg and Lee (2017) for more details.

ROBUSTNESS CHECKS

In the main text of our paper, we used a common hold-out method for testing such that we performed cross-validation on the training data and out-of-sample testing on the hold-out data. To ensure the robustness of our analysis, we use a cross-validation method for testing to ensure that our model generalizes well to unseen data. In particular, we use a 10-fold cross-validation for testing. The dataset was randomly split into 10 folds, and the model was trained and tested 10 separate times such that each fold got a chance to be the test data and the other 9 folds were used as training data. The results are summarized in the table below, which show consistent performance. This is likely due to the stability and advantage of a large dataset in our study. When the dataset is small, the results are more likely to be sensitive to different train-test splits.

Table W2 Out-of-Sample Performance for Predicting $Y\_complete$ Using 10-fold Cross-Validation

Fold	**#1**	**#2**	**#3**	**#4**	**#5**
Accuracy	92.12%	92.03%	92.07%	92.12%	92.16%
Fold	**#6**	**#7**	**#8**	**#9**	**#10**
Accuracy	92.28%	92.17%	92.11%	92.26%	92.22%

In our dataset each instructor only has one course. To examine whether the results are driven by the person specific fixed effects, we repeat the prediction analysis for with or without the course-level fixed effects (e.g., adding or removing course ID dummy variables). The results are reported in the table below, suggesting that, even though brands/instructors may play an

important role when enrolling consumers, once the consumers have purchased access to the course, their consumption behavior for each video within the course mostly depends on the video content itself (note that the instructor does not change from video to video within the same course, only the content does). In other words, consumers are likely to purchase an online course because of their interest in the brand/instructor. But once they start watching video lectures in that course, their video consumption behavior will be mainly driven by the content given that the brand/instructor is the same for each and every portion of the video.

Table W3 Out-of-Sample Performance for Predicting $Y\_complete$

	Accuracy	Precision	Recall
With Person Specific Fixed Effects	0.92	0.89	0.93
Without Person Specific Fixed Effects	0.92	0.89	0.93

In the main text of our paper, we used a nonparametric approach called gradient boosting machine (Friedman 2001) to predict consumer behavior. In the two tables below, we show that other machine learning algorithms such as random forests (Breiman 2001) achieve similar performance.

Table W4 Out-of-Sample Performance for Predicting $Y\_complete$

	Accuracy	Precision	Recall
Gradient Tree Boosting	0.92	0.89	0.93
Random Forests	0.92	0.89	0.93
AdaBoost	0.92	0.89	0.93
SVM	0.92	0.89	0.92
Logistic Regression	0.77	0.75	0.69

Table W5 Out-of-Sample Performance for Predicting Y_next

	Accuracy	Precision	Recall
Gradient Tree Boosting	0.88	0.88	0.86
Random Forests	0.87	0.87	0.86
AdaBoost	0.88	0.88	0.86
SVM	0.87	0.87	0.86
Logistic Regression	0.84	0.83	0.85

One unique characteristic of our data is that the video segments are relatively short (average video length = 11.6 minutes). Unlike movies or television shows, videos on MasterClass usually do not have an ending sequence with the full cast and crew information. Instead, MasterClass instructors usually talk until the last second in video lectures. Therefore, to acquire full knowledge of the video, users need to watch the entire video. In the two tables below, we show that our results are robust to using a less strict definition of video completion and watching next video.

Table W6 Out-of-Sample Performance for Predicting $Y\_complete$

	Accuracy	Precision	Recall
Threshold = 99.9%	0.92	0.89	0.93
Threshold = 99%	0.90	0.89	0.90
Threshold = 98%	0.90	0.88	0.90

Table W7 Out-of-Sample Performance for Predicting Y_next

	Accuracy	Precision	Recall
Threshold = 5%	0.88	0.88	0.86
Threshold = 4%	0.87	0.87	0.87
Threshold = 3%	0.87	0.87	0.88

In our prediction analysis, the out-of-sample prediction is based on new consumers. That is, the training/test split is conducted at the consumer level (e.g., shuffling consumer ID), and consumers in the test data have never appeared in the training data. We have conducted another out-of-sample prediction using new courses (e.g., shuffling course ID) so that course videos in the test data have never appeared in the training data. We have also added another out-of-sample prediction using both new courses and new consumers (e.g., shuffling both course ID and consumer ID). That is, neither course videos or consumers in the test data have appeared in the training data. Both results are reported in the table below. These results show that the prediction performance for new courses (accuracy = 0.76) and new courses with new consumers (accuracy = 0.75) are lower than that of new consumers (accuracy = 0.92).[3] This is expected because predicting brand new courses is a more difficult task. However, we also note that, although the prediction accuracy of predicting new courses is lower than that of predicting new consumers, it is still significantly higher than the average baseline 60% (Recall the average completion rate was 0.397 as shown in Table 3 in the paper).

[3] In a robustness check where we remove consumers' past viewing data, the results show that our proposed framework achieves an average out-of-sample prediction accuracy of 74.50% when predicting video competition.

Table W8 Out-of-Sample Prediction Performance

	Accuracy	Precision	Recall
Predict New Consumers	0.92	0.89	0.93
Predict New Courses	0.76	0.75	0.76
Predict New Courses and New Consumers	0.75	0.74	0.75

To explore how different features vary across different videos within a course, we have added two sets of descriptive statistics for the two different courses as presented in the table below. We note that for those features that do not vary much within a course (e.g., hair color, facial hair, makeup, glasses), they are indeed shown to have little predictive power as shown in the individual-level feature importance analysis in the paper.[4]

[4] We thank an anonymous referee for this suggestion.

Table W9 Summary Statistics for Two Different Courses

	Course #1		Course #2	
Variable	**Mean**	**SD**	**Mean**	**SD**
Basic Video Properties:				
Chapter	4.497	2.886	9.368	7.219
Video Length	11.251	4.568	13.116	4.781
Average Scene Length	0.089	0.016	0.351	0.136
Speaking Rate	143.450	19.568	118.676	17.316
Sentiment	0.144	0.072	0.029	0.024
Instructors' Emotions and Physical Characteristics:				
Anger	0.006	0.005	0.023	0.023
Contempt	0.005	0.004	0.002	0.005
Disgust	0.003	0.002	0.012	0.006
Fear	0.002	0.004	0.000	0.001
Happiness	0.345	0.134	0.197	0.086
Neutral	0.577	0.111	0.735	0.095
Sadness	0.037	0.021	0.025	0.015
Surprise	0.024	0.021	0.005	0.016
Age	31.354	0.806	68.034	2.458
Gender	0.000	0.000	1.000	0.000
Glasses	0.119	0.323	0.074	0.262
Facial Hair	0.000	0.001	0.106	0.014
Baldness	0.319	0.121	0.321	0.049
Hair Color	3.676	1.882	3.924	0.473
Makeup	1.000	0.000	0.008	0.062
Smile	0.345	0.134	0.197	0.086
Visual Aesthetic Features:				
Foreground Motion Area	0.456	0.072	0.142	0.044
Motion Magnitude	1.180	0.348	0.542	0.138
Motion Direction	3.140	0.047	3.078	0.096
Warm Hue Proportion	0.438	0.163	0.498	0.130
Saturation	0.307	0.022	0.219	0.043
Brightness	0.469	0.038	0.195	0.030
Contrast of Brightness	0.229	0.012	0.171	0.011
Clarity	0.996	0.005	0.976	0.037

When converting videos to lower dimension, we use a commercial video converter software named Wondershare Video Converter Ultimate which compresses video with little quality loss. In a robustness check we also show that converting videos to a lower resolution level at 640*360 pixels still preserves representative information of the original video. The results are shown in the tables below. Moreover, we also attach two full screenshots below for a more straightforward illustration for the high-quality video conversion. We can see that although the details in low-resolution video is not as clear as those in high-resolution video, the low-resolution video is still of high quality which preserves similar distributions of pixels in the original video.

Table W10 Comparing High-Resolution and Low-Resolution Versions for the Same Video

	High Resolution	**Low Resolution**
Warm Hue Proportion	0.640	0.639
Saturation	0.296	0.292
Brightness	0.374	0.374
Contrast of Brightness	0.120	0.120
Clarity	0.870	0.870

A. Screenshot for Original High-Resolution Video

B. Screenshot for Low-Resolution Video after Conversion

Figure W5 Screenshots Comparison

DIFFERENT MACHINE LEARNING ALGORITHMS

When predicting students' viewing behavior with video genome features, we use different machine learning models to corroborate the results. Each of these models can be implemented using `scikit-learn` (Pedregosa et al. 2011). Some descriptions of the algorithms in this appendix are adapted from https://scikit-learn.org/.

Gradient Tree Boosting

Gradient boosting machines are nonparametric models that draw a parallel between boosting and gradient descent in function space (Friedman 2001). They additively build up simple models, where each successive model is built by predicting the residuals of the preceding model to increase model performance. We use a regression tree which is the most common type of basic model to predict the residuals. Gradient Tree Boosting considers additive models of the following form:

$$F(x) = \sum_{m=1}^{M} \gamma_m h_m(x)$$

where $h_m(x)$ are the basis functions (e.g., decision trees), which are usually called weak learners in the context of boosting. Decision trees have a number of abilities that make them valuable for boosting, namely the ability to handle data of mixed types, and the ability to model complex functions. Similar to other boosting algorithms, Gradient Tree Boosting builds the additive model in a greedy fashion:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

where the newly added tree h_m tries to minimize the loss L, given the previous ensemble F_{m-1}:

$$h_m = arg\min_h \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h(x_i)).$$

Gradient Boosting attempts to solve this minimization problem numerically via steepest descent. The steepest descent direction is the negative gradient of the loss function evaluated at the current model F_{m-1} which can be calculated for any differentiable loss function:

$$F_m(x) = F_{m-1}(x) - \gamma_m \sum_{i=1}^{n} \nabla_F L(y_i, F_{m-1}(x_i))$$

where the step length is chosen using line search:

$$\gamma_m = arg\min_\gamma \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) - \gamma \frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}).$$

AdaBoost

AdaBoost, short for Adaptive Boosting, is a machine learning meta-algorithm that aims to convert a set of weak classifiers into a strong one to improve performance (Freund and Schapire 1997). The output of the weak classifiers is combined into a weighted sum that represents the final output of the boosted classifier, which can be represented as

$$F(x) = sign\left(\sum_{m=1}^{M} \theta_m f_m(x)\right)$$

where f_m stands for the m^{th} weak classifier and θ_m is the corresponding weight. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. The individual learners can be weak, but as long as the performance of each weak learner is slightly better than random guessing, the final model can be proven to converge to a strong learner.

In each boosting iteration, the data modifications consist of applying weights $\omega_1, \omega_2, \ldots, \omega_N$ to each of the training samples. Initially, those weights are all set to $\omega_i = 1/N$, so that the first step simply trains a weak learner on the original data. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the reweighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that were missed by the previous ones in the sequence (Hastie et al. 2009).

Random Forests

A random forest constructs a multitude of decision trees at training time and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees (Breiman 2001). It can correct for decision trees' tendency to overfit the training set. Random forests combine the general technique of bootstrap aggregation and random selection of features to construct a collection of independent decision trees with controlled variance. In

particular, each tree in the ensemble is built from a sample drawn with replacement from the training set. When splitting each node during the construction of a tree, the best split is found either from all input features or a random subset. The purpose of these two sources of randomness is to decrease the variance of the forest estimator.

SVM

Support Vector Machine (SVM) is a widely used classification algorithm which finds the optimal hyperplane to separate positive and negative classes while leaving the largest possible margin on both sides of the decision boundary. Specifically, the SVM classifier works by solving the following optimization problem:

$$\min_{w_p, b_p} \frac{1}{2} w_p^T w_p + C \sum_{i=1}^{N} \xi_i$$

$$\text{s.t.} \quad y_i\left(w_p^T x_i + b_p\right) \geq 1 - \xi_i, \qquad \xi_i \geq 0, i = 1, \ldots, N_p,$$

where w_p and b_p are model parameters of the maximum-margin hyperplane and ξ_i are slack variables that penalize data points that violate the margin requirements.

CODE EXAMPLES

When generating video genome features, we use several different methods from machine learning and computer vision to quantify relevant variables from each video such as average scene length and emotions. In this Web Appendix, we briefly describe those methods and demonstrate how they are implemented in Python.

Average scene length is calculated as the average amount of time between scene cuts, which is based on an intelligent scene cut detection algorithm implemented using `PySceneDetect` (https://pyscenedetect.readthedocs.io/). We first use a content-aware scene detection algorithm to

detect scene changes in each video. The content-aware scene detector works the way most people think of "cuts" between scenes in a movie: given two frames, do they belong to the same scene, or different scenes? The content-aware scene detector compares each frame sequentially looking for changes in content, which is useful for detecting quick cuts between scenes based on detecting consecutive frames in a video that have significant differences. The following code sample illustrates the general workflow to perform scene detection programmatically. It provides an example for detecting the scenes on an input video (`testvideo.mp4`) and printing the scenes to the terminal/console. We have also included more code examples for detecting instructors' emotions and physical characteristics as well as motions in the videos in our data.

```
from __future__ import print_function
import os

import scenedetect
from scenedetect.video_manager import VideoManager
from scenedetect.scene_manager import SceneManager
from scenedetect.frame_timecode import FrameTimecode
from scenedetect.stats_manager import StatsManager
from scenedetect.detectors import ContentDetector

STATS_FILE_PATH = 'testvideo.stats.csv'

def main():
    # Create a video_manager point to video file testvideo.mp4. Note that
multiple
    # videos can be appended by simply specifying more file paths in the list
    # passed to the VideoManager constructor. Note that appending multiple
videos
    # requires that they all have the same frame size, and optionally,
framerate.
    video_manager = VideoManager(['testvideo.mp4'])
    stats_manager = StatsManager()
    scene_manager = SceneManager(stats_manager)
    # Add ContentDetector algorithm (constructor takes detector options like
threshold).
    scene_manager.add_detector(ContentDetector())
    base_timecode = video_manager.get_base_timecode()

    try:
        # If stats file exists, load it.
        if os.path.exists(STATS_FILE_PATH):
            # Read stats from CSV file opened in read mode:
            with open(STATS_FILE_PATH, 'r') as stats_file:
```

```
                stats_manager.load_from_csv(stats_file, base_timecode)

        start_time = base_timecode + 20     # 00:00:00.667
        end_time = base_timecode + 20.0     # 00:00:20.000
        # Set video_manager duration to read frames from 00:00:00 to
00:00:20.
        video_manager.set_duration(start_time=start_time, end_time=end_time)

        # Set downscale factor to improve processing speed.
        video_manager.set_downscale_factor()

        # Start video_manager.
        video_manager.start()

        # Perform scene detection on video_manager.
        scene_manager.detect_scenes(frame_source=video_manager)

        # Obtain list of detected scenes.
        scene_list = scene_manager.get_scene_list(base_timecode)
        # Like FrameTimecodes, each scene in the scene_list can be sorted if
the
        # list of scenes becomes unsorted.

        print('List of scenes obtained:')
        for i, scene in enumerate(scene_list):
            print('    Scene %2d: Start %s / Frame %d, End %s / Frame %d' % (
                i+1,
                scene[0].get_timecode(), scene[0].get_frames(),
                scene[1].get_timecode(), scene[1].get_frames(),))

        # We only write to the stats file if a save is required:
        if stats_manager.is_save_required():
            with open(STATS_FILE_PATH, 'w') as stats_file:
                stats_manager.save_to_csv(stats_file, base_timecode)

    finally:
        video_manager.release()

if __name__ == "__main__":
    main()



# Detecting Instructors' Emotions and Physical Characteristics
import requests
import json

# set to your own subscription key value
subscription_key = None
assert subscription_key

# replace <My Endpoint String> with the string from your endpoint URL
face_api_url = 'https://<My Endpoint String>.com/face/v1.0/detect'
```

```
image_url =
'https://upload.wikimedia.org/wikipedia/commons/3/37/Dagestani_man_and_woman.
jpg'

headers = {'Ocp-Apim-Subscription-Key': subscription_key}

params = {
    'returnFaceId': 'true',
    'returnFaceLandmarks': 'false',
    'returnFaceAttributes':
'age,gender,headPose,smile,facialHair,glasses,emotion,hair,makeup,occlusion,a
ccessories,blur,exposure,noise',
}

response = requests.post(face_api_url, params=params,
                         headers=headers, json={"url": image_url})
print(json.dumps(response.json()))



# Detecting Motions
import numpy as np
import cv2 as cv
cap = cv.VideoCapture(cv.samples.findFile("vtest.avi"))
ret, frame1 = cap.read()
prvs = cv.cvtColor(frame1,cv.COLOR_BGR2GRAY)
hsv = np.zeros_like(frame1)
hsv[...,1] = 255
while(1):
    ret, frame2 = cap.read()
    next = cv.cvtColor(frame2,cv.COLOR_BGR2GRAY)
    flow = cv.calcOpticalFlowFarneback(prvs,next, None, 0.5, 3, 15, 3, 5,
1.2, 0)
    mag, ang = cv.cartToPolar(flow[...,0], flow[...,1])
    hsv[...,0] = ang*180/np.pi/2
    hsv[...,2] = cv.normalize(mag,None,0,255,cv.NORM_MINMAX)
    bgr = cv.cvtColor(hsv,cv.COLOR_HSV2BGR)
    cv.imshow('frame2',bgr)
    k = cv.waitKey(30) & 0xff
    if k == 27:
        break
    elif k == ord('s'):
        cv.imwrite('opticalfb.png',frame2)
        cv.imwrite('opticalhsv.png',bgr)
    prvs = next
```

References

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 119-139.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 1189-1232.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems (NeurIPS)* (pp. 4765-4774).

Molnar, C (2018). Interpretable machine learning - a guide for making black box models explainable.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct), 2825-2830.

Shapley, L. S. (1953). A value for n-person games. *Contributions to the Theory of Games*, 2(28), 307-317.

Štrumbelj, E., & Kononenko, I. (2014). Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41(3), 647-665.