Advanced Analytics Assignments

D0S07a

Team Members:

Ivo L. Arasin - r0926378
Vancesca Dinh - r0830510
Linas J. Leščinskas - r0874301
Yavuz Yavuzhan - r0920104

May 27, 2023

Contents

Associated files, scripts and pictures for all four assignments are to be found here https://github.com/vtdinh13/advanced-analytics-projects

1. Assignment 1

1.1 Exploratory Data Analysis

The aim of this report is to detail the approach our group has taken in an effort to build a predictive model for per-night prices of AirBnB listings. The dataset contains 6,495 property listings in Belgium, primarily in the city, and includes 55 features. Features include mixed data types (e.g., textual, numerical, dates, geospatial). Although the dataset came preprocessed to a certain extent, many features are not ready to be used in the modeling stage. Transformations of various data types and imputations of missing values are required.

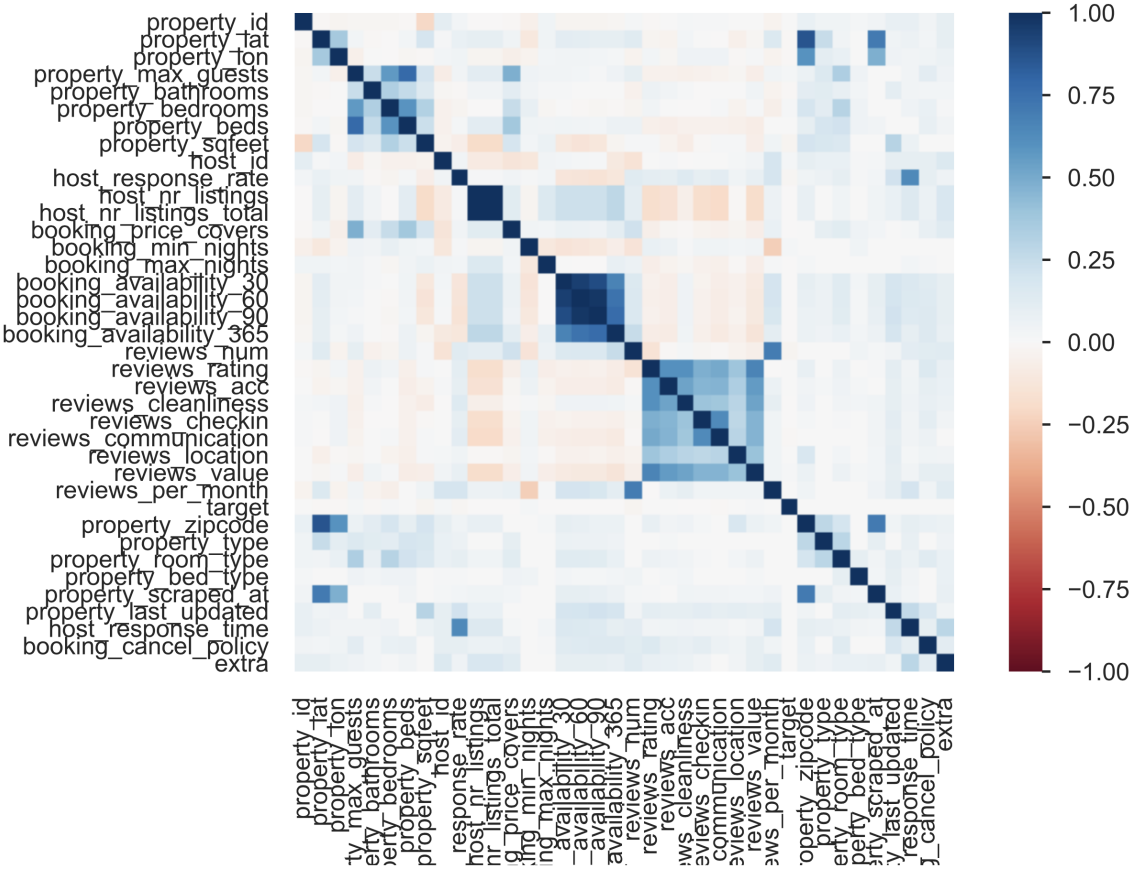


Figure 1.1: Correlation heatmap of all original features

Initial exploratory data analysis revealed that, at least up front, no variable has any explanatory power with respect to the target (c.f. Figure 1.1). 34 features have missing values, although this is counting all features that have at least one observation missing. Still, 21 features have more than 10% missing values. The worst offender is `property_sqfeet`

with more than 97% of its values missing. Many of the missing values, especially in the reviews-category coincide (c.f. Figure 1.2). Additionally, there are many textual features that would require transformation to be useful. There are no duplicate rows.

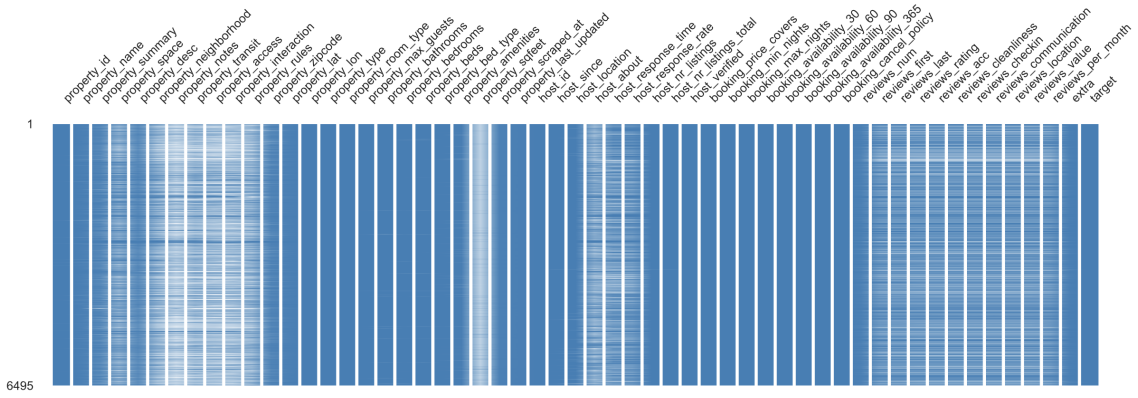


Figure 1.2: Nullity matrix of all original features

In what follows, we detail the steps taken to preprocess and manipulate the data to distill as much predictive power as possible.

1.2 Data Cleaning & Feature Engineering

Missing values mostly coincide in the `reviews` category(c.f. Figure 1.2). However, this does not happen in all cases. The inclusion of features other than reviews remedies this shortcoming.

1.2.1 Numerical features

Numerical features with missing values are imputed with linear regression whose independent variables were picked based on observed correlation with the feature to be imputed (e.g., `reviews_acc` is imputed based on `host_nr_listings`, `booking_availability_365`, `reviews_rating`, `reviews_num`, `reviews_value`, `reviews_cleanliness`, `reviews_checkin`, and `reviews_location`.

1.2.2 Geolocation features

`property_sqfeet`, despite its many missing values, might hold some information. A closer look reveals grave inconsistencies in the observations that do exist. Sometimes the values indicate square-feet, other times square-meters. Visual inspection of square-feet and target price reveals that a good threshold to differentiate between the two metrics is 150 (based on trial and error and the intuition that there are no listings offering a room with a size of $< 14m^2$). Accordingly, all listings with `property_sqfeet` < 150 are multiplied with the conversion factor 10.764 to convert m^2 to $feet^2$. Missing values were replaced with zeros. When filtering out all zero-values, `property_sqfeet` now has a correlation with the target of $\sim$23%, as opposed to the $\sim$15% before.

Missing `property_zipcode` values are imputed based on k-nearest-neighbours with n=4. Then, the `property_zipcode` column is replaced with two new columns. One indicates the frequency of occurrence and the other mean-encodes the target per zipcode(i.e., for every zipcode-group the target-mean is computed based on the training data). If the test

data contains previously unseen zipcodes, the column makes use of the overall target-mean computed based on the training set.

The coordinates as given by `property_lat` and `property_lon` are transformed from Cartesian to polar coordinates. Later, a density based clustering is applied to segment all observations into 50 clusters.

Density based clustering of coordinates using DBSCAN

Since in every city, there are variations in socioeconomic makeup between neighborhoods, with some areas being wealthier and others offering more affordable housing options, density based clustering is applied. In the hopes of capturing some of those differences, first the coordinates as given by *property_lat* and *property_lon* are transformed from cartesian to polar coordinates, then the DBSCAN algorithm is implemented for both cities (Brussels and Antwerp) separately. The algorithm implements density-based clustering using polar coordinates and detects concentrated zones across the city. It does not take into account target prices during the calculation and is based only on the train-set, thereby ensuring the *Chinese Wall* between the train-set and test-set is never breached.

The algorithm is applied iteratively by varying parameters, namely ϵ and minimum number of points. Starting from a low ϵ and high sample size requirement, higher density areas are captured first, and requirements to be considered as a cluster are relaxed in next steps. In the end, 47 clusters are identified for Brussels and 5 clusters for Antwerp.

The map of Brussels after clustering can be seen in Figure 1.3. The Folium package is used to visualize and explore geospatial data throughout our study. Lastly, distance from city centers for each observation is calculated for both cities, however it did not result in significant improvements to the predictions and is ignored in the final model.

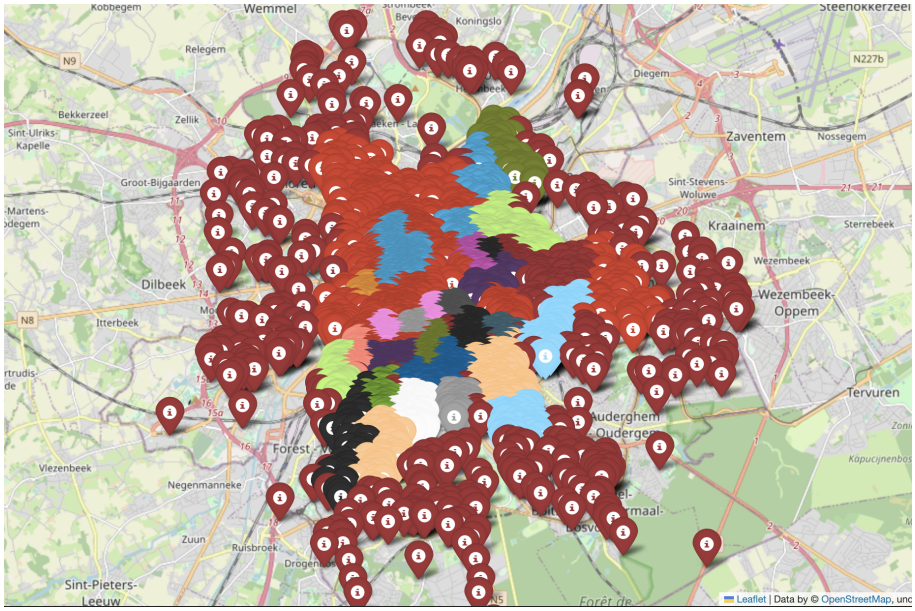


Figure 1.3: DBSCAN clusters of Brussels

1.2.3 Text-based features

In an effort to extract information out of text-based features, a pre-trained sentiment classifier is applied to indicate the sentiment of descriptions. We hypothesize that descriptions might vary in the extent to which they praise their own listing. This might translate

to predictive power. We apply the sentiment classifier "nlptown/bert-base-multilingual-uncased-sentiment" taken from Hugging Face to the text-based features provided. Ultimately though, the text-based features could not improve the performance of any of the models we chose; therefore, we decide to drop them.

1.2.4 Array-features

Some features contain an entire array of different items with variable length per observation. Such features include *property_amenities*, *host_verified* and *extra*. For all of these, all unique items observed in the training-set in any of the arrays are extracted and transformed into a dummy-column. For the model itself, the majority of the dummy-columns are dropped, except those that improve explanatory power, as determined by lowering the root mean square error (RMSE) value on the validation-set.

Amenity-features such as "Hangers", "Family/kid-friendly" and "24-hour check-in" turn out to be useful (at least for the validation set). For `extra` (e.g., "Host Has Profile Pic" and "Host Identity Verified") appears to hold some explanatory power. For `host_verified`, not a single item changes the RMSE value much, so it is omitted entirely. We also attempted to derive additional features from existing ones (e.g. the number of verification methods, amenities, etc.), but these derived features do not improve RMSE.

1.2.5 Categorical features

Features such as `property_type` are inspected with respect to the target and binned anew into fewer, more homogenous classes. Subsequently, they are turned into an ordinal feature where the order is according to the variance observed in the target per class. However, in our train-validation-test split, this feature has no explanatory power and is thus omitted.

1.2.6 Time features

The feature `property_last_updated` is transformed into a numerical feature where text is turned into the corresponding number of days (e.g. class "a week ago" corresponds to 7, "2 months ago" is made to be 60 days, "a day ago" is mapped to 1, etc).

The features, `reviews_first` and `reviews_last`, are aggregated and replaced by a new feature indicating the number of days in between the first and last review. This feature seems to hold high explanatory power relative to all others.

`host_response_time` is transformed into a simple ordinal feature with higher integer-numbers representing a longer response time.

1.2.7 Outliers

It is apparent that there are many outliers in the data set. Isolation Forest is used as a way to calculate the anomaly score of all observations. The parameters included are the following: `host_nr_listings`, `reviews_num`,`booking_availability_365`, `reviews_rating`, `reviews_cleanliness`, `reviews_checkin`, `reviews_location`, and `reviews_value`.

After some fine-tuning, a good value for the contamination-hyperparameter (i.e. the percentage of data to be classified as anomalous) is set to 4%. All anomalous instances are dropped and not considered any further. Many of those are extreme in the target, are located far outside the two cities of Brussels and Antwerp, have suspiciously high values for `booking_price_covers`, have an unreasonably high demand for the minimum number of nights to be booked, or have not been updated in recent time.

Crucially, the outlier removal is done just before training the model, not before imputing other features and doing feature engineering (e.g., mean-encoding zipcodes). Despite

seemingly somewhat unorthodox, this chronology appears to result in overall better performance when compared to no outlier removal or outlier removal before imputation and feature engineering.

1.3 Variable Selection

All pre-processing and feature engineering described pertains only to features ultimately used in the model, except when explicitly stated otherwise. Variable selection is based on whether inclusion reduces the RMSE of the validation set and its feature-importance (c.f. Figure 1.4) as determined by the number of splits in the `XGBoost` model. Table 1.1 summarizes the list of features used in the final model.

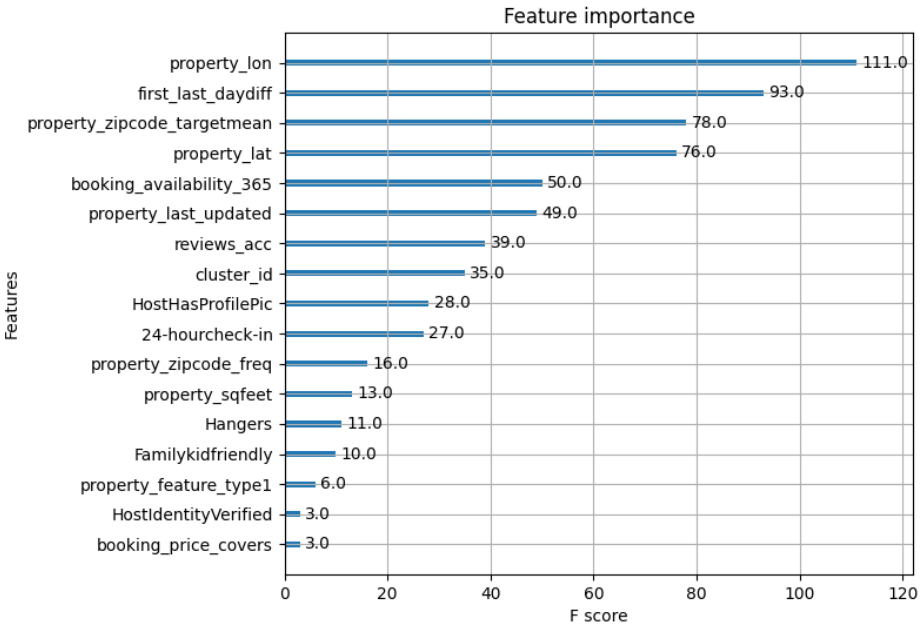


Figure 1.4: Feature importance (calculated by the number of times a variable was split on)

1.4 Predictive Model

RandomForest, XGBoost, and linear regression were tested as candidate models. Linear regression is tested but excluded from any further analysis as it is immediately outpaces by the ensemble techniques considered. RandomForest almost achieves parity with `XGBoost` but always ever so slightly lags behind. Thus, the clear method of choice remains `XGBoost`.

As a baseline score for orientation, we calculate the mean (67.9) of the entire dataset and use it as a single-value predictor. The corresponding RMSE value on the test-set is 46.718 and the Mean Average Error (MAE) is 28.138. Throughout the process of trying out various models and manipulating the data back and forth, both scores prove elusive and difficult to reach, suggesting generally little predictive power in our models.

After fine-tuning XGBoost on our validation-set, we settle on a maximum-tree-depth of 3, a subsampling-ratio of 0.8, a minimum-child-weight of 5 and a learning-rate of 0.02.

Table 1.2 summarizes various scores for the validation- and test-set. Our model corresponds to tuned XGBoost. The test-metrics table indicates both the normal and the

Feature	Treatment
property_lat	Cartesian to polar coordinates
property_lon	Cartesian to polar coordinates
property_zipcode_freq	engineered feature: frequency-encoded
property_zipcode_targetmean	engineered feature: mean-encoded
property_feature_type1	existing feature re-binned
property_feature_type3	existing feature re-binned
first_last_daydiff	engineered feature: difference in days
reviews_acc	kept as is, missing values imputed
booking_price_covers	kept as is
property_last_updated	kept as is, text variables mapped to integer equivalents
cluster_id	engineered feature: density-based clustering
Hangers	extracted feature from property_amenities
Familykidfriendly	extracted feature from property_amenities
24-hourcheck-in	extracted feature from property_amenities
property_sqfeet	existing feature cleaned and transformed
booking_availability_365	kept as is
HostHasProfilePic	extracted feature from extra
HostIdentityVerified	extracted feature from extra

Table 1.1: List of features used in final model

		RMSE	MAE
Validation metrics			
	Tuned XGBoost	45.434	28.353
	Mean	45.435	28.855
	Random Forest	45.680	30.021
	Untuned XGBoost	57.138	33.063
Test metrics			
	Mean	46.718	28.138
	Tuned XGBoost	47.045	27.523
	Hidden Mean	58.89	31.344
	Hidden tuned XGBoost	59.328	30.687

Table 1.2: Performance Metrics

hidden score, reflecting two separate test-sets of which only one was visible during model construction. Since testing a new set of predictions was not possible after the publication of the hidden test-scores, the hidden mean is inferred by comparing teams' previous publicly visible score to their hidden score (thus it is not entirely certain).

1.5 Conclusion

Table 1.2 clearly demonstrates that the best "model" in terms of predictive performance is the arithmetic mean, highlighting the likely possibility the data holds vanishingly little predictive power.

Furthermore, it is evident there are extreme values in the test-set as the RMSE value of our model is larger than that of the mean-predictor while our MAE value is smaller than that of the mean-predictor (in both the public- and the hidden-test sets). This indicates that while our model has some, albeit marginal, explanatory power, it only manages to capture

the presence of extreme observations insufficiently well. On our validation-dataset, however, our model manages to slightly outperform the mean-predictor (admittedly after fine-tuning the model based on the validation-set, thereby incorporating it in our training-set to an extent).

A maximum tree-depth of 3 is prohibitive in the sense that only relatively coarse classification is possible. That this depth is the optimum in our case is another indication the data does not hold much predictive power. Figure 1.5 plots our predictions against the real-values of the validation set. The plot also shows the extent to which the data is beset with extreme values(i.e., listings with very high or low target prices).
Overall, building a predictive model based on the data at hand proved astoundingly difficult. Despite various revisions and approaches, our model could not convincingly beat the arithmetic mean.

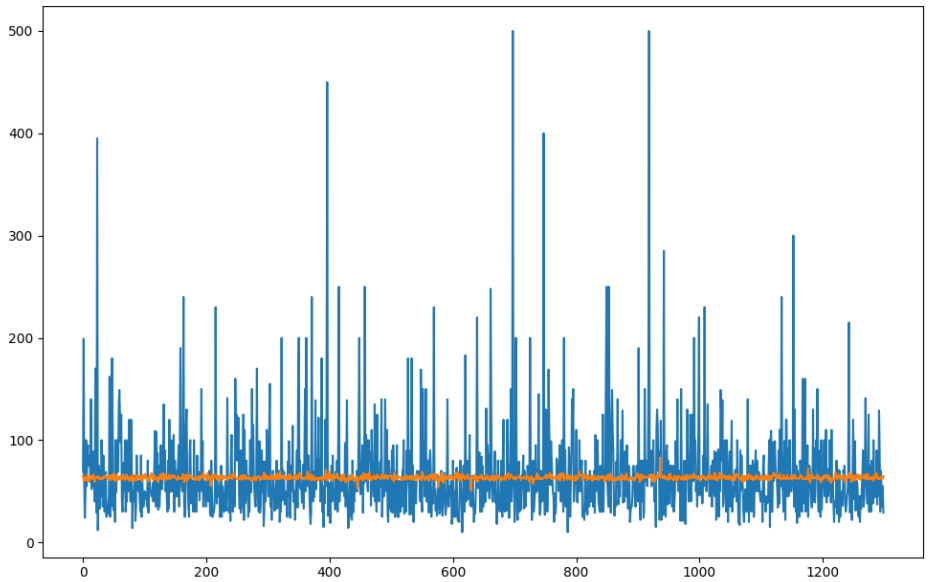


Figure 1.5: Plot of predicted values (orange) and targets (blue) on validation set

2. Assignment 2

2.1 Introduction

The dataset at hand comprises nearly 120k images of the *Guide Michelin* to restaurants around the world. The majority is evidently made up of pictures of food-items and dishes. Secondly, a good third is of the restaurant interiors. Finally, a marginal portion is of undefined class since they can show housefronts, gardens, the chefs themselves, some intricate piece of decor or anything else. The task is to classify these images according to a defined set of labels.

We take on a two-pronged approach and tackle two tasks that complement one another. First of all, we built a model to sort the images into three classes, namely *dish*, *interior* and *other* where *other* corresponds to anything that does not strictly fall within either the *dish* or the *interior* class (task C). In a second step, we use the identified dish-images to built a model sorting them into the largest 20 cuisine-types (task A). Since the development of both models is independent of one another, we first elaborate on the first model (Task C) and in the second chapter of this report explicate the cuisine-classifying model (Task A).

2.2 Task C

2.2.1 Transfer Learning

Given the size of the dataset and the complexity of the task, designing, building and training a convolutional neural network from scratch represents an outsized effort since we can draw on *transfer learning* where an existing, already trained model is used as a basis to start with. Generally, such pre-trained models boast a carefully designed architecture, are trained on reams of data and on a vast number of classes. As a consequence, one can assume the earlier layers all the way up to the head of the neural network have already learned some general properties and generalized distinctive features that can be quickly leveraged by replacing the top-layers with untrained ones and retraining the transplanted part on the desired dataset while keeping the weights and biases of the rest untouched. This allows for generally good results with minimal data, time and compute.

2.2.2 Model set-up

We decided to use *Xception*, a convolutional neural network 71 layers deep, trained on the ImageNet dataset with 1000 classes. After loading the model, we proceed by removing the last dense layer representing the 1000 classes along with the preceding global average pooling layer without trainable weights, replacing it with a global-average-pooling layer followed by a dense layer of three neurons for the three classes. A softmax-activation function is used for the prediction layer. In total, the model has 20,867,627 parameters of which 6,147 are trainable while the remainder is left frozen. Input images are of size

256x256x3 and are loaded in batches of size 16. A train-validation split of 80% and 20% is set up. Python and the tensorflow-based deep-learning library Keras are used.

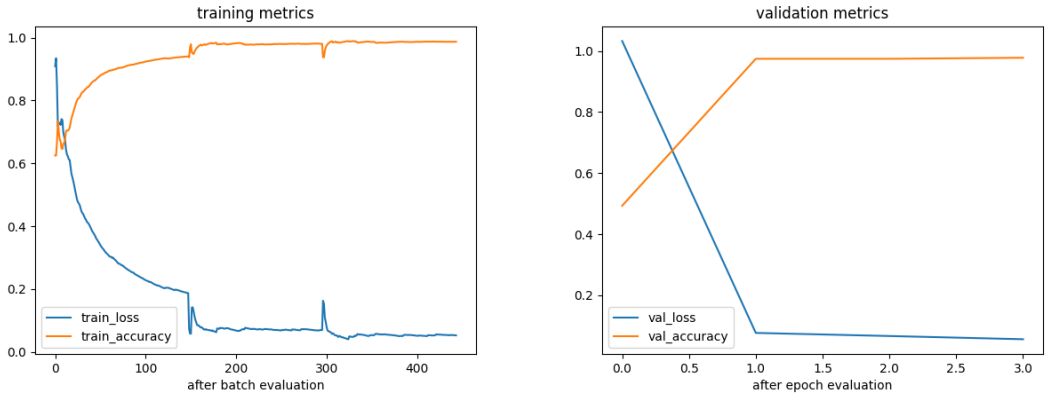
2.2.3 Data Labelling and Augmentation

Since for our task the data was not yet labelled, about 500 images were labelled manually into the three classes *dish*, *interior* and *other*. Our interpretation of *interior* captures any type of space where patrons of a restaurant may sit while dining, thus the term itself is not to be interpreted in a strict sense and may also refer to outdoor spaces with relevant furniture. Once done, a first model was expediently trained to aid in the further labelling-process. Ultimately, the training set comprises about 2400 images, of which 1700 belong to the class *dishes*, 700 to *interior* and about 100 to *other*. As this presents an acute class-imbalance, data augmentation is used on the *other*-class. Augmentation entails randomly adjusting brightness levels between ±45%, contrast levels between ±25% and rotating images within a range of ±10%.

Afterwards, the dataset contains about 2900 instances with about 500 belonging to *other*.

2.2.4 Training

The model is trained for three epochs as more seem to lead to overfitting but in the first three accuracy still improves throughout, albeit marginally. Thus, a form of early-stopping is applied to avoid unwanted model properties. As is apparent from below Figure 2.1, training just with one epoch already leads to a very good accuracy score of >97%. However, the loss seems to still decrease significantly from the first to the last epoch indicating the two additional epochs were worthwhile. Once fully trained, the model achieves a validation-loss of 0.0558 and a validation-set accuracy of 97.79%.



(a) Training Loss and Accuracy

(b) Validation Loss and Accuracy

Figure 2.1: Evaluation metrics observed during training. *Keras* does not provide an interface to register validation metrics after evaluation on one batch, thus the short x-axis. The apparent spikes in the training metrics at the start of every new epoch appear due to the train-set size not being divisible by the batch size into an integer, slightly distorting metric-calculation.

2.2.5 Interpretability: Integrated Gradient

Once trained, it is difficult to ascertain if the model is picking up on globally salient features as a result of which it assigns correct labels. One visually intuitive technique is the integrated gradient which allows to contrast the activation of each individual pixel between a target-image and one of randomness, called the baseline (black and white are also often used). It does so by approximating and aggregating the the change per pixel in terms of its gradient over a number of interpolation-steps between the baseline image and the target image.

Ideally, once the per-pixel activation is visualized, it should resemble certain shapes or patterns that are present in the target image due to which the model then manages to attribute a correct label. If this isn't the case, despite the label being correct, one might wonder why the model's assigned label was correct and it can be indicative of sensitivity, lack of general interpretability or something else.

We've applied the integrated gradient to multiple images on our trained model to investigate if humanly comprehensible features were extracted and are used as a decision-making basis. Figure 2.2 depicts three pictures with one of each class (i.e., *interior*, *dish*, and *other*). The pictures suggest the model did indeed learn genuine properties of the objects and spaces it is supposed to classify. That is most acutely visible in image (a) where the highlights trace the rim of the plate and its contents.

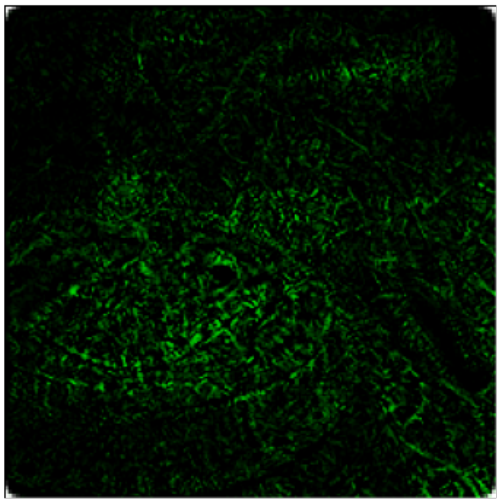
For the interior (c) it is less striking, yet when observing multiple iterations it is clear that the model delineates the wine-glasses.

Least convincing is image (e), although windows and the entrance are dimly contrasted along with the tree on the right-hand side and the general square shape of the mansion is crudely captured.

This behaviour makes sense, though, as the underlying model is trained on the ImageNet dataset and is thence innately able to recognize objects such as glasses, plates, cutlery which are strongly associated with the first two categories our model is supposed to classify but not with the "lump"-category *other*. Still, given the overall accuracy of 98% there is no pressing need to fine-tune or further tweak our model as all other categories next to *interior* and *dish*, which seem to come naturally to the pre-trained model, are comprising the *other* category and are consequentially easily separated.

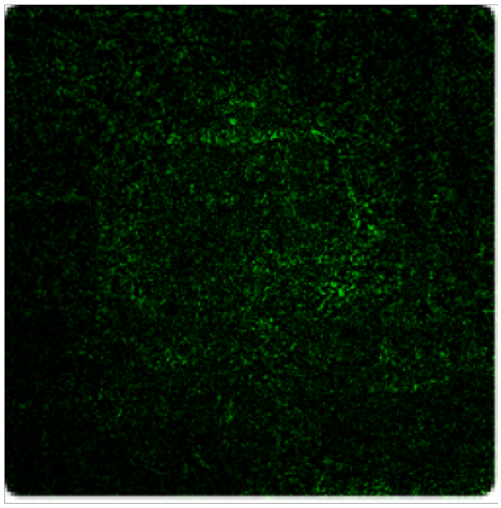
It is also worth mentioning that, depending on the baseline-image used for the integrated gradient method, the highlighted features are least robust, i.e. most varied, for the class *other* whereas for the other two classes, the visualized integrated gradient is very robust and does not change much when comparing black, white or random noise baseline images. This, once more, reflects the aforementioned point.

(a) Image of class *dish*



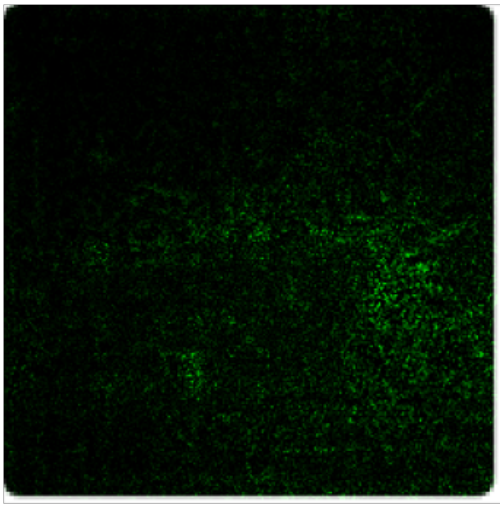
(b)

(c) Image of class *interior*



(d)

(e) Image of class *other*



(f)

Figure 2.2: Integrated Gradient

2.2.6 Model Evaluation

The model manages to reliably classify images into the three classes. Figure 2.3 displays a set of set of test-images which the model has neither seen in the training-set nor in the validation-set yet has successfully classified. Notice, however, image (i) and image (n) where the prediction is not fully confident. With image (n) the inherent ambivalence of the differentiation between *interior* and *other* might be reflected. Often, certain types of spaces such as gardens or the driveway leading up to a restaurant can not easily be told apart from a place's interior where people dine if, for example, the dining place is within the garden or outdoors. Some ambivalence is introduced as a result.

Below table summarizes the model's performance on a test-set of 505 unseen images. The model achieves an accuracy of 97.6%.

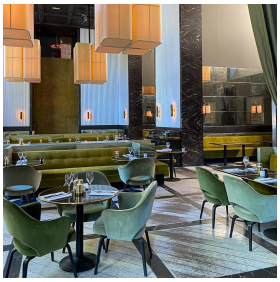
	prediction			
truth	dish	interior	other	*recall*
dish	351	0	0	100%
interior	1	101	3	96.19%
other	1	7	42	84%
precision	99.43%	93.52%	93.33%	**97.63%**

(a) *dish* 99.9%

(b) *interior* 99.2%

(c) *other* 99.4%

(d) *dish* 99.9%



(e) *interior* 99.3%

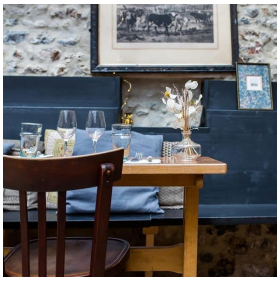
(f) *other* 99.5%

(g) *dish* 99.4%

(h) *interior* 99.3%

(i) *other* 55.2%; *interior* 46%

(j) *dish* 98.0%



(k) *interior* 99.8%

(l) *other* 93.1%

(m) *dish* 99.9%

(n) *interior* 77.5%; other 21%

(o) *other* 96.8%

Figure 2.3: Model predictions on unseen images (class, probability)

2.3 Task A

Since the first model manages to sort images into the three categories of *dish*, *interior* and *other* rather reliably, this second model will now be trained only on dish-images. As the cuisine-classifying model has a far higher, much more granular and distinct set of classes unique to itself, a larger dataset is needed (as compared to the model in Task C where about 2400 images were enough to attain good accuracy). Transfer learning is also likely to be less effective as compared to the first model because the classes models such as *Xception* are trained on do not readily map or correlate with the highly specialized and peculiar factors characterising different types of haute-cuisine (at least not as much as they do with the classes used in Task C). First, we simplified some of the cuisine classes e.g. merged *Modern French*, *Contemporary French*, *Creative French* to one class named *French*. Then, we have chosen to work with only top-20 most represented classes - of about 90.000 images belonging to the largest 20 cuisine-types, roughly 44,000 images of dishes are identified using the first model and are used as input for this task. It is important to note that most frequently occurring cuisine class *Modern Cuisine* is dropped due to computational limitations.

2.3.1 Data exploration

Subsequently, the most frequent class in our dataset used for Task A is *Creative* (Figure 2.4). It comprises roughly 14% of the images in the dataset. Consequently, our aim is to construct the model that is able to classify images with considerably higher accuracy than simply always picking the most frequent class.

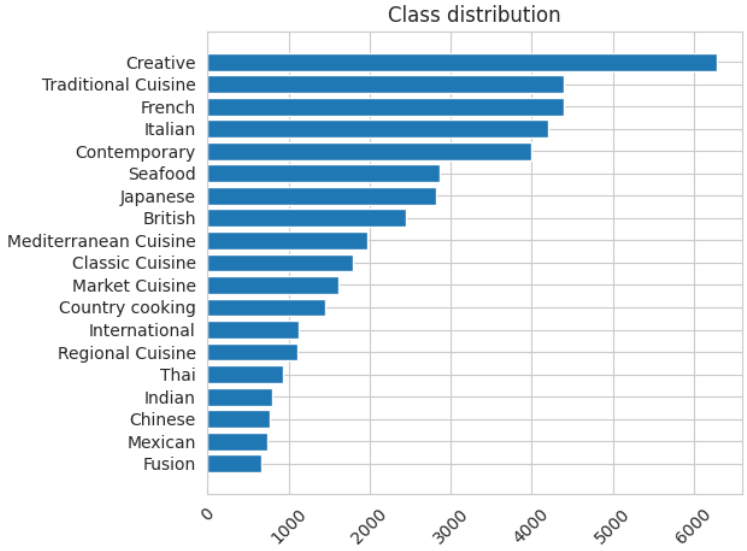


Figure 2.4: Class distribution for classification Task A

Next, we inspect some images together with corresponding labels in the dataset (Figure 2.5). It is evident that certain cuisines are hardly distinguishable even for a human eye. For example, pictures labeled *British* and *French* look almost identical. In addition, either of them could be labeled *Creative* or *Contemporary*. To complicate matters further, additional inspection of sample images from the dataset convinced us that differentiating between *Creative* and *Contemporary* or *Classic Cuisine* and *Market Cuisine* is virtually impossible, at least for non-experts in the field.

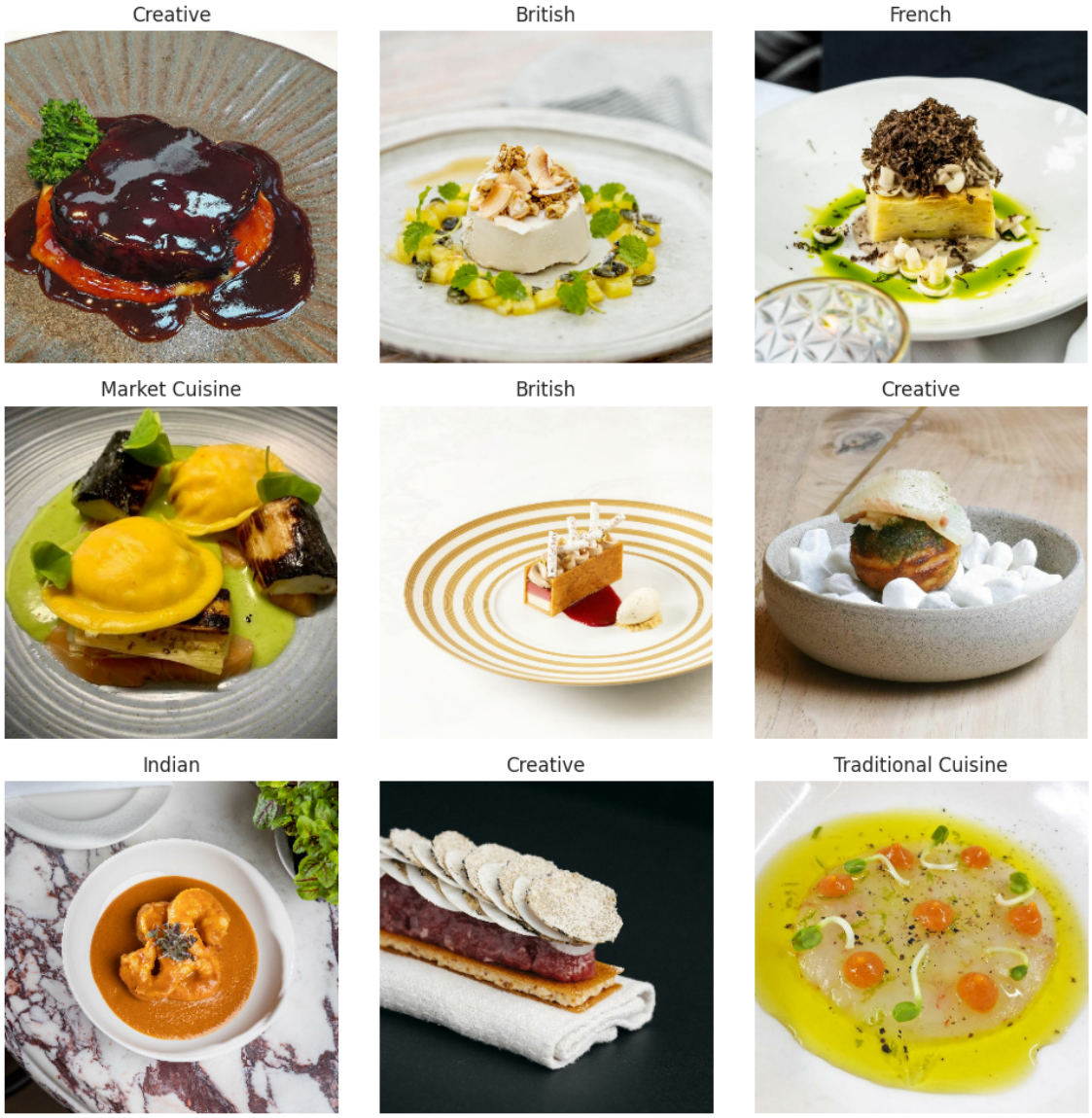


Figure 2.5: Some images and their corresponding labels from the dataset

Hence, we are skeptical about extracting considerable discriminatory performance from the model. In other words, "what is not had, cannot be given".

2.3.2 Model set-up

Here we utilize the same pre-trained model (*Xception*) as for Task C. We proceed by peeling off the last layer, replacing it with a global-average-pooling layer followed by a dense layer with 256 neurons to capture non-linear patterns in the considerably complicated and intricate image data. We also include a dropout layer to induce regularization and finish with a dense layer of 19 neurons, corresponding to the number of classes in the task. A softmax-activation function is used for the final prediction layer. In total, the model has 21,390,907 parameters of which 529,427 are trainable while the remainder is left frozen. Input images are of size 256x256x3 and are loaded in batches of size 16. A train-validation split of 80% and 20% is used.

2.3.3 Data augmentation

To extract as much predictive performance as possible, we apply image augmentation to the training dataset. We chose to modify images by introducing random flipping, rotation, zoom, translation and contrast. Figure 2.6 displays some augmentations for a random image in a training set.

Figure 2.6: Example of image augmentation

2.3.4 Training

The model is trained on three epochs as training the model on 35,500 images is both a computationally- and time-intensive task. As is apparent from Figure 2.7, training accuracy is improving only slightly from the first epoch onwards. The model simply learns to guess the most frequent class (*Creative*) and is unable to extract actual discriminatory performance. Similarly for model loss, it is rapidly going down throughout the first epoch, but then converges to a local minimum and is unable to decrease further. Once fully trained, the model achieves a validation-loss of 2.7351 and a validation-set accuracy of 14.11%.

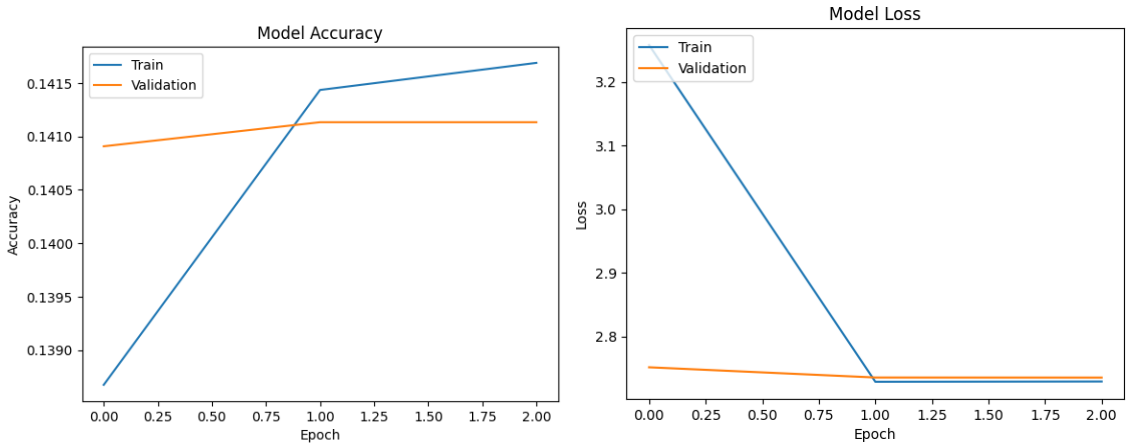


Figure 2.7: Accuracy, Loss for Training and Validation

2.3.5 Model Predictions

After inspecting model predictions on unseen images (Figure 2.8), it is apparent that the model simply learned to always guess the label *Creative*. As already mentioned, images in the dataset may not be distinctive enough to achieve any notable classification performance. However, hyper-parameter tuning, a higher number of epochs and more unfrozen layers during model training process could potentially yield some performance gain. Nevertheless, given that our model represents a simple but reasonable first approach to this arguably hard problem, one would expect at the very least a performance superior to guessing the most frequently occurring class. Since this is not the case, we are hesitant about further exploration with larger, more complicated models both due to limited available compute and acute time-constraints.

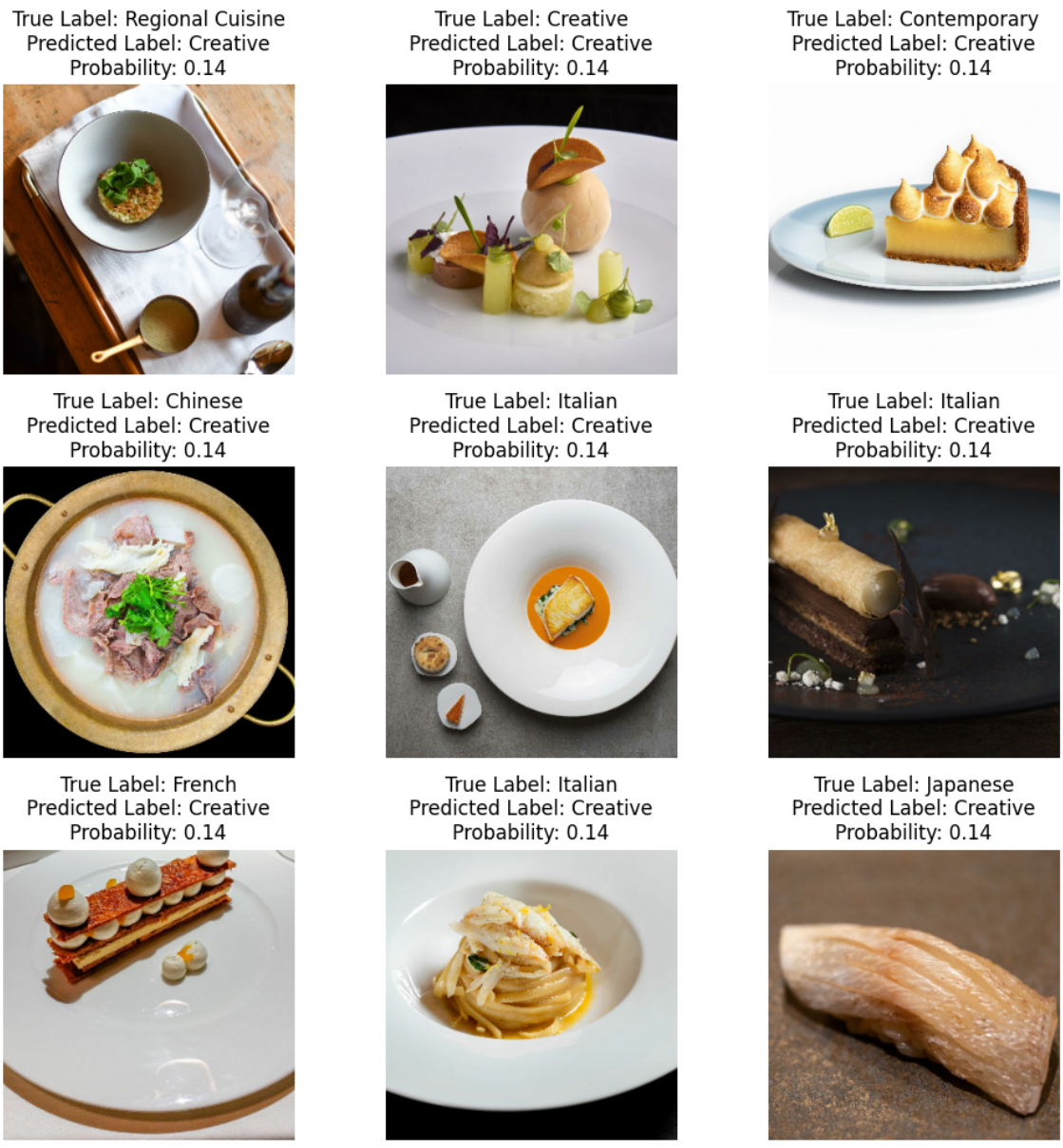


Figure 2.8: Model predictions on unseen images

3. Assignment 3

3.1 Introduction

The goal of this assignment is to set up Spark and work in a streaming environment. This section briefly details the approach our group has taken to work with streaming data and is structured according to three key tasks of this assignment.

3.2 Predicting on streamed textual data

3.2.1 Construct a data set using the provided stream

Multiple group members took the effort of collecting the streaming data from the provided endpoint. We use 250 non-empty batches to construct the data set.

First of all, we define an empty dataframe (constructed from empty RDD) with a schema to match the structure of JSON files that are received via streaming:

```
# Define empty RDD
empty_rdd = spark.sparkContext.emptyRDD()
# Define schema for the dataframe
columns = StructType([StructField('review_id', StringType(),
    False),
StructField('app_id', StringType(), False),
StructField('review_text', StringType(), False),
StructField('label', StringType(), False)])
# Create empty dataframe
df = spark.createDataFrame(data=empty_rdd, schema=columns)
df.show()
```

Then, we utilize a simple for-loop to go through directories and read in the received JSON files that contain Steam reviews. Some simple checks are performed to remove duplicate rows (in case several team members received the same batches) and remove empty reviews. Eventually, a data set that contains 793 rows is constructed. The class distribution is rather skewed, with 647 'positive' reviews and 146 'negative' reviews.

3.2.2 Construct a predictive model to predict the review target based on the review text

We split the data set into train and test portions, where 30% is set aside for testing the model. Then we define a simple pipeline.

First, we separate words into tokens with RegexTokenizer, using any non-word character as a delimiter. Then, we remove stopwords with StopWordsRemover. In addition, we asked ChatGPT to generate a list of commonly used slang words in Steam reviews and appended that list to the one generated by the class. Then we consider two featurization approaches:

- **CountVectorizer**: to transform filtered words (tokens) into a matrix of token counts.
- **TF-IDF features**: to obtain a vector of weights that rewards frequently occurring words in a data set. We do not use external corpus to penalize frequently occurring words in a data set that also occur frequently in a given corpus.

Finally, we fit the logistic regression model (twice to consider both approaches). Model with CountVectorizer achieves 83% accuracy, whereas TF-IDF model performs slightly worse, registering accuracy of 81%. We save both models to a local directory so that they can be utilized for predictions as the stream comes in.

3.2.3 Use trained model to show you can make predictions as the stream comes in

In this step we adapt the provided notebook (*spark_streaming_example_predicting.ipynb*) to use the trained models for 'live' prediction of sentiment as the stream comes in. Essentially, we load the model from a local directory and apply it to a dataframe constructed from incoming JSON files. Then we select only relevant columns to display.

Here we exhibit code and output of the model with CountVectorizer features:

```
# Add capability to predict as the stream comes in
globals()['models_loaded'] = False
globals()['my_model'] = None

def process(time, rdd):
if rdd.isEmpty():
return

print("=========␣%s␣=========" % str(time))

# Convert to dataframe
df = spark.read.json(rdd)
df.show()
# Load in the model if not yet loaded:
if not globals()['models_loaded']:
# load in your models here
globals()['my_model'] = PipelineModel.load('/home/linas/
   Desktop/kul/$em␣2/advanced␣analytics␣in␣business/
   assignment␣3/model/')
globals()['models_loaded'] = True
# Make predictions with loaded model
df_result = globals()['my_model'].transform(df)
df_result.select("app_id", "label", "review_id", "
   review_text", "prediction").show()
```

We can see that model successfully outputs predictions on the incoming stream (Figure 3.1). In addition, it is able to predict the majority of the sentiments correctly. Hence, our entire pipeline is functioning as intended.

Needless to say, the model still trips on some instances. That is to be expected as it demonstrated imperfect accuracy on the test set as well. The model could be improved by constructing a larger data set and considering more sophisticated featurization and modeling approaches. However, achieving superior predictive performance is not the main goal of this assignment.

```
+-------+-----+---------+--------------------+
| app_id|label|review_id|         review_text|
+-------+-----+---------+--------------------+
| 669330|    1|138534240|Loads of cool des...|
| 669330|    1|138533323|Very fun it has a...|
|2369390|    0|138534914|Extraordinarily b...|
|2369390|    1|138534604|Vote me as new El...|
|2369390|    1|138534371|You're the Lucky ...|
|2369390|    1|138534307|fun game with all...|
+-------+-----+---------+--------------------+


+-------+-----+---------+--------------------+----------+
| app_id|label|review_id|         review_text|prediction|
+-------+-----+---------+--------------------+----------+
| 669330|    1|138534240|Loads of cool des...|       1.0|
| 669330|    1|138533323|Very fun it has a...|       0.0|
|2369390|    0|138534914|Extraordinarily b...|       0.0|
|2369390|    1|138534604|Vote me as new El...|       1.0|
|2369390|    1|138534371|You're the Lucky ...|       1.0|
|2369390|    1|138534307|fun game with all...|       1.0|
+-------+-----+---------+--------------------+----------+
```

Figure 3.1: Example output of predicting on streamed reviews

3.3 Challenges

Throughout the work on this assignment, we encountered several challenges that are briefly detailed below.

- Spark was generally very slow. It took 60 minutes to fit the pipeline on the training set and another 90 minutes to compute the accuracy of the model. Setting Spark to work with 4 cores (instead of 1 core as default) improved the situation. However, it was evident that Spark is not optimized to run on a single machine i.e. locally.
- The stream to receive reviews kept continuously crashing after a handful of minutes while collecting raw data, necessitating a restart of the notebook cell. Multiple group members were receiving the batches to alleviate the effort.

4. Assignment 4

4.1 Preface

Twitch offers a pool richly endowed with content wide in breadth and gluttonous in depth. The task at hand is to take on a small sliver of said pool in an open ended analysis. In this report, we delve into a hypothetical scenario where we assume the role of a data analytics consulting company. The question frame is as follows:

> *An advertising firm has approached us to help them identify the "ideal candidate" for targeted marketing on Twitch. Our task is to find ideal candidates defined by quantitative measures that will yield the highest return on investment.*

In an effort to solve this problem, we seek to identify the characteristics that make streamers popular. Our approach involves exploring various characterising metrics about streamers and clusters. We draw on simple aggregate statistics, modularity for clustering, random walks for a measure of viewer-retention and borrow ideas from the text-mining domain, specifically tf-idf and Latent Dirichlet Allocation (LDA), in an effort to describe clusters and unearth additional insights.

4.2 Data

In the exploratory stage, we observe that the number of recommendations a streamer accumulates does not necessarily correspond to the number of average views a streamer receives. This means that while a streamer may have a large number of recommendations, this does not necessarily translate to a large number of views. Going forward, we will look at the number of average views as the main criterion for measuring the ideal candidate.

We decide to work on a subset that contains all streamers with average views of at least 1100 along with their relations and related nodes (i.e. `squads`, `tags` and `games` are kept). The remainder is discarded without further consideration. This results in 4681 streamers, representing a little more than 5% of the total amount of streamers in the original dataset. From an advertiser's perspective, this is sensible since ads should be placed to those with large audiences. More generally, it is likely that more data would merely corroborate existing, but not bring to light fundamentally new insights. This, however, is something which we have only anecdotally verified but certainly not proven. Furthermore, any attempt to visualize such voluminous data will inevitably result in a clutter that is more artistic than revealing.

Additionally, we decided to omit `squads` from further analysis. Inspection of the relationship between streamers and squads in a network graph shows many microscopic and isolated clusters containing at most a handful of streamers. A few are thinly connected by one streamer belonging to multiple clusters. Ultimately though, we believe this is not germane to overall clustering efforts. We also only consider `tags` that appear at least ten times. Thus, our analysis henceforth focuses on a dataset that includes: 4681 streamers, 2476 games, 1024 tags, and 77573 directed relations between them.

4.3 Analysis

To cluster the data, modularity is computed in *Gephi* using the Louvain method. It detects 34 clusters. As many of the clusters are rather small, we only consider those with a size of at least 5% of total streamers (i.e. a cluster should contain at least 234 streamers) for more detailed analysis. ForceAtlas2 is used for the graph layout.

It is noteworthy to point out that even though modularity and the graph layout algorithm of ForeAtlas2 do not communicate with one another (i.e. both methods were executed independently of one another), the visually discernible clusters appear to strongly correlate with the node-color, as determined by modularity. Thence they complement each other very well.

4.3.1 Betweenness

To enrich the visualization, the betweenness score is calculated (based on *recommends*-relationships) for each streamer and used as the node-size. Larger nodes correspond to nodes with high betweenness. As could be expected, streamers with high betweenness mostly sit at the edge or between clusters, thus acting, in-effect, as a "hub" (c.f. Figure 4.1).

From an advertiser's perspective, streamers with high betweenness are of great interest because they are assumed to have access to more than just the community they identify with, thus having the potential to reach a wider audience. This is of great value to advertisers looking to promote their products or services to a large and diverse group of people.

4.3.2 Cluster characterization

Borrowing a method from text-mining, Latent Dirichlet Allocation (LDA) is applied on a per cluster basis to analyse associated games and tags.

Each streamer within a cluster is linked to a number of games (or tags). For each cluster, all linked games (or tags) are aggregated and compose the representative vector. This vector is just a long list of all games (or tags) played, where duplicates are of course allowed. Such a representation can be converted into a bag-of-words representation with tuples indicating which game is played and how often it is played (which tag occurs and how frequent it is). Said bag-of-words vector essentially corresponds to a document composed of different words. All clusters together form the corpus.

LDA is applied individually both on cluster-documents composed of games and on cluster-documents composed of tags. For both, the number of topics is set to 20 (remember, number of clusters is 34) to induce some abstraction and not rely just on a frequency count per cluster (which would be the case if the number of topics was set to 34).

All clusters are linked to one topic only with a probability of >90%. Figure 4.1 depicts the clustering along with the clusters' most descriptive games and tags.

4.3.3 Evaluation

As could be expected, a handful of highly popular games dominate all clusters, such as "Just Chatting", "Grand theft Auto V", "VALORANT", "League of Legends" etc. Furthermore, dominant languages such as English can also be found in every cluster. Still, in spite of this and the notable degree of dispersion of the blue (31) and orange (14) cluster, all clusters are distinct in some way.

The green (17) community on top is mainly composed of Russian speakers and popular games are "Dota 2", Virtual Casino" and "Counter Strike: Global Offensive". The dark

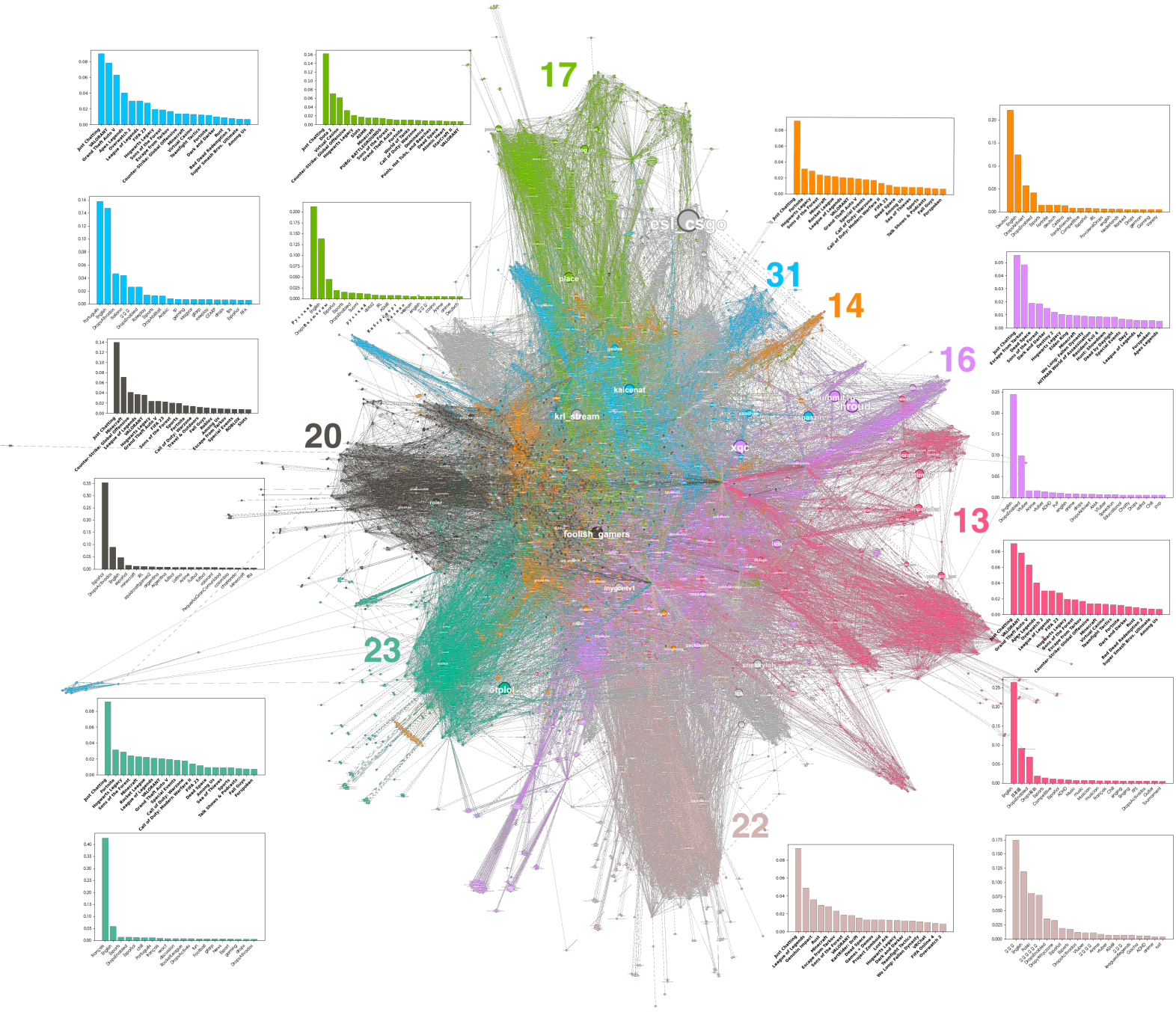


Figure 4.1: Streamers and games in graph network, streamer-node sized based on betweenness. Graphs show LDA-topics of games (bold) and tags associates with clusters. See appendix for details.

grey community (20) is composed predominantly of Spanish speakers with the most popular game being "Minecraft" (not considering "Just Chatting" as a game). The ming-green cluster (23) is French in majority and likes to play "Fortnite".

It is also notable that clusters are differentiated in large part by language, indicating that language is an important feature for Twitch's recommendations.

4.4 Analysis beyond clustering

4.4.1 PageRank and Markov Chain

[1]Our client is seeking to identify valuable customer segments in order to optimize their limited budget and maximize their benefits. We propose that the strategy should focus on increasing exposure and fostering consistent engagement with a specific set of customers. In other words, rather than solely maximizing the overall viewership of their advertisements, our hypothesis posits maintaining high visibility and continuous interaction with the target audience is crucial. To achieve this goal, we aim to identify distinct communities where both viewers and streamers exhibit loyalty and tend to remain within the same cluster. If some of the streamers within a community are chosen for an ad, their viewers should actively navigate between other streamers in the community but stay within the circle. For measuring this behavior, we make the following assumption: *Viewers tend to explore other streamers they encounter through recommendations.*

Then, we apply the Markov Chain and Pagerank algorithms to assess the modularity of communities and evaluate the disjointness of recommendation edges. Our findings reveal that certain communities exhibit promising potential for targeted marketing, as they demonstrate limited outgoing edges beyond their respective communities. The methodology we employ consists of the following steps:

1. Implement the Louvain algorithm on a reduced dataset, encompassing all types of nodes, to extract the communities.
2. Create a transition/adjacency matrix for the communities by aggregating the recommends edges. The transition matrix is a square matrix of order N, with values converted to probabilities by dividing them by the sum of the corresponding row.
3. Apply the Markov Chain transition for 20, 50, and 100 steps, and store the results.
4. For each community, generate a subgraph comprising its streamers and apply the Pagerank algorithm to identify the top 5 most visited streamers.
5. For each community, generate a personalized Pagerank vector for the top 5 streamers, assigning equal probabilities, and set the remaining streamers' probabilities to 0. The assumption here is that viewers will explore the recommended channels and occasionally return to the top streamers.
6. Run the personalized Pagerank algorithm for all communities in the entire graph.
7. Aggregate the probabilities by communities to determine the percentage of nodes that remained within the community compared to those that ventured outside.
8. Combine the results from Pagerank and the Markov process. Include statistics such as median views_avg and the average number of followers in the communities.

[1]This part of the analysis is done on a separately made clustering, which is why the given cluster IDs in this section do not correspond to the other cluster IDs found in section 4.3. Since the data is the same, however, insights remain perfectly valid.

9. Sort the communities based on their retention score and median view count, selecting candidates that meet a predefined threshold level and have a sufficient number of streamers.

10. Analyze the communities and deduce their characteristics by examining the associated tags, games, and squad information.

4.4.2 Illustrative insights

:

- *Chess players (community 13)*: Even though they are a smaller community, they are closely connected with a random walk retention probability with 0.98. They also have more than 2000 median view count. They are predominantly chess enthusiasts but they also play "Rocket League". These people also have the tag of "Dead by Daylight", indicating that they are night owls.

- *Korean indie-game players (community 15)*: They have the top performance in Markov-chain retention, even in the 100 period case, indicating that they have loyal and closely-knit viewer base. These communities demonstrate relatively lower scores in terms of degree centrality for tags and games, suggesting an inclination towards novelty and a willingness to explore new experiences. Their viewers might be a good fit for a new product launch as a potential of early adopters.

- *Minecraft and Sports players who are Spanish in majority (community 5)*: They form a community of about 500 streamers and have a random walk retention probability of 0.99 and Markov-chain transition probability of 0.84 in 20 period-case. Their median view-count is a little over 2000. They are also recommended by other communities therefore viewer exposure could be further increased.

Such insights can be valuable when wanting to target specific demographics. The three examples above are not exhaustive but give an indication that homogeneous communities can be identified in accordance to any advertisement-requirements.

Additionally, upon analyzing the results, we once again find that the recommendation algorithm tends to favor language-based groupings among streamers. In other words, streamers are more likely to be recommended from other streamers who speak the same language and share similar tags. A tabular overview can be found in the appendix with table 4.4.

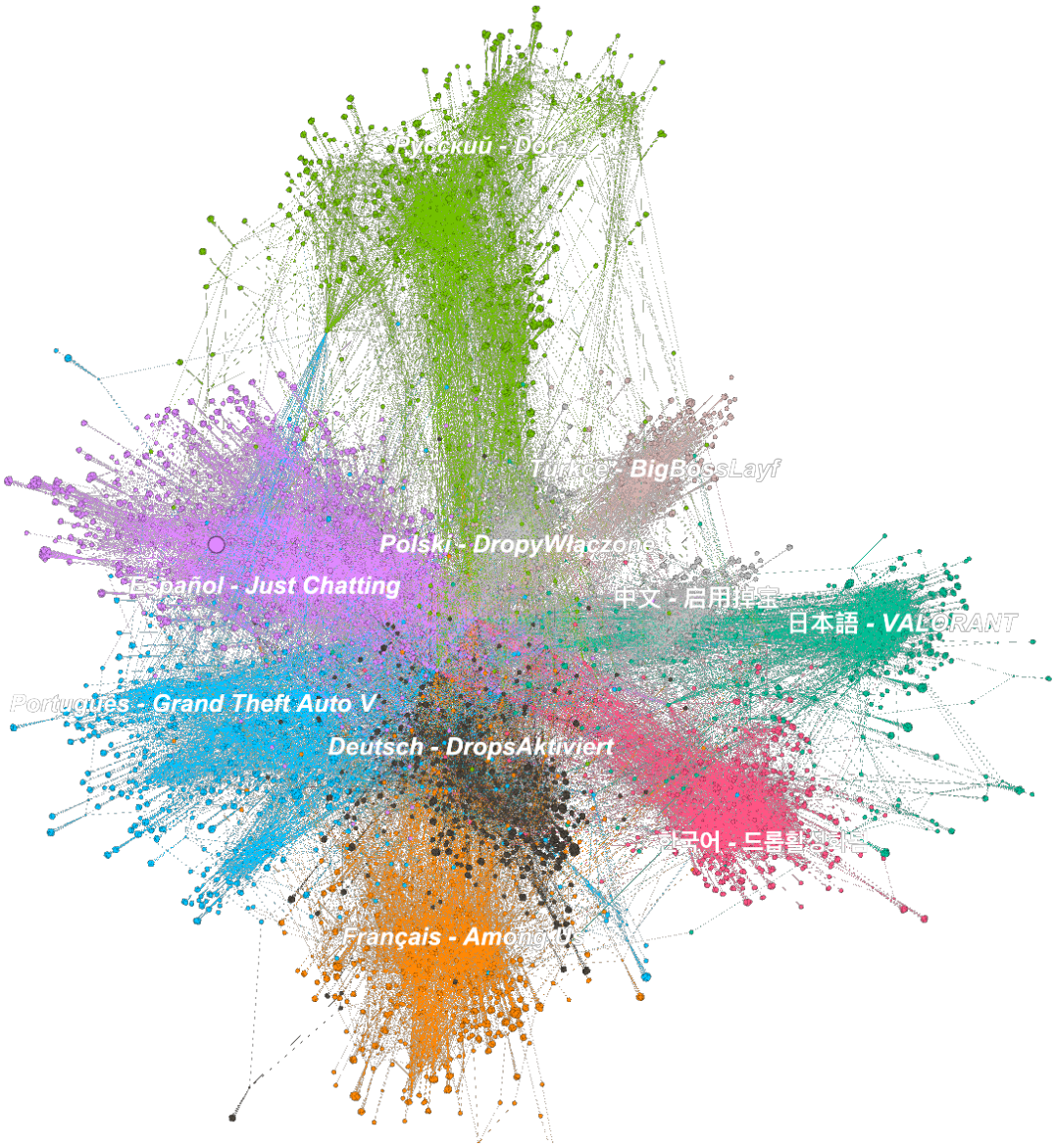


Figure 4.2: Top 10 communities with highest retention scores

4.4.3 tf-idf

Although betweenness lends itself quite naturally to a clustering set-up, many other approaches exist. An advertiser might not be interested in the bridge-building behaviour of streamers but in some measure of their uniqueness. We define uniqueness as streamers that are popular (i.e. all those within our subset since they have at least 1100 average views) but do not play generally popular games and are thus catering to a hardened nerd-audience or just happen to dominate some domain of games which others neglect. The concept of tf-idf from text-mining is well suited to represent this metric. If we think of individual games as words and of streamers who play those games as composite documents thereof, we can readily apply tf-idf. Those that play games many others are playing too will be punished (inverse-document frequency) whereas those that play games that no-one else is streaming will gain. Streamers who do both will have mid-ranged tf-idf scores (c.f. Figure 4.3).

For example, "wolviehd" is streaming a single game and has a high tf-idf score as it is a game no one else is playing. The flock of streamers streaming only "Just Chatting", which is the overall most popular "game", correspondingly have a minimal tf-idf. Another case in point is the streamer "gamesdonequick" who streams a large number of games but still has a relatively high tf-idf score, or uniqueness as the games played are almost entirely exclusive to this streamer.

As an advertiser this property of streamers adds an additional degree of granularity and can be used for specific targeting. An indie-game developer might find it interesting to place ads alongside "gamesdonequick" (in pink cluster 16) due to the streamer's large audience, high degree of differentiation, or uniqueness, and special focus on niche-games.

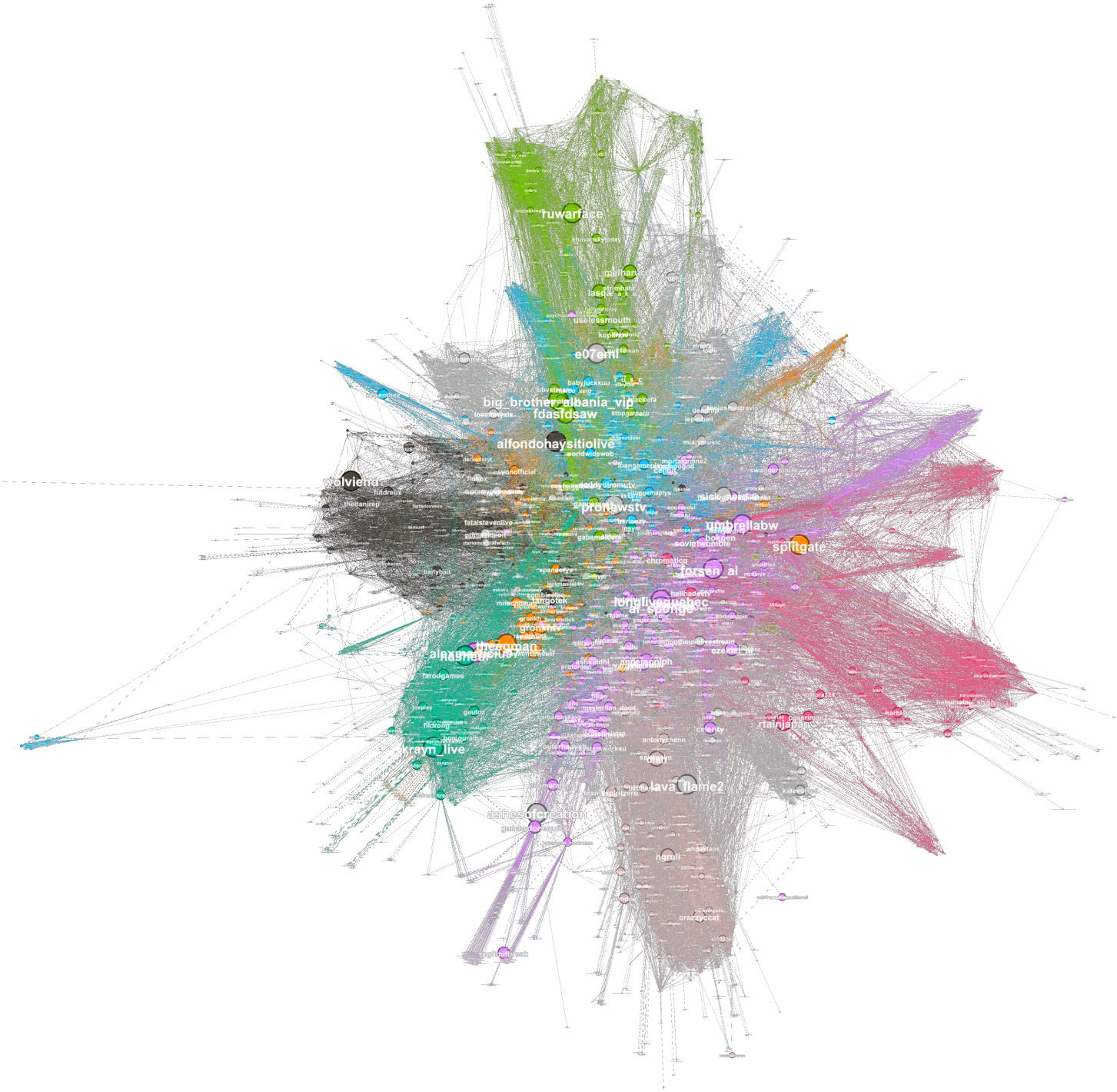


Figure 4.3: Streamers and games in graph network, streamer-node sized based on tf-idf

4.5 Concluding remarks

For a brief illustration of our finding's applicability, let us assume an advertiser tries to identify a suitable target for the purpose of promoting a new e-sports themed energy drink. Firstly, the language of a cluster will be considered. Afterwards, the cluster can be further broken down into games that are interesting for the ad to be served. VALORANT in cluster 13 is a good target given its association with e-sports (see list of tags). Then, a suitable streamer must be selected, preferably one with high reach. This might be "valorant_emea", based on the relatively high betweenness score. Once selected, the choice can be further refined by the cluster's retention or by the streamer's uniqueness score. In the current case, a low uniqueness might be preferable since the advertiser is keen to reach a homogeneous group where all viewers enjoy the same games. "valorant_emea"'s uniqueness-score is indeed 0.0019. With an average viewership of >18.000, a good target was effectively identified.

Ultimately, an exhaustive analysis seems to be out of reach but was not the aim of us anyway. The data undoubtedly still holds much more that what we have brought to light which is why we loosely framed our analysis from the perspective of an advertiser to make sense of findings and direct our efforts broadly in one consistent direction. Non the less, our discoveries do reveal a good degree of differentiation among clusters. This dissection is insightful from a standpoint of curiosity but becomes actually valuable when inspected from the vantage-point of an advertiser. Measures such as the Random Walk or the tf-idf score then allow for additional nuance.

4.6 Appendix

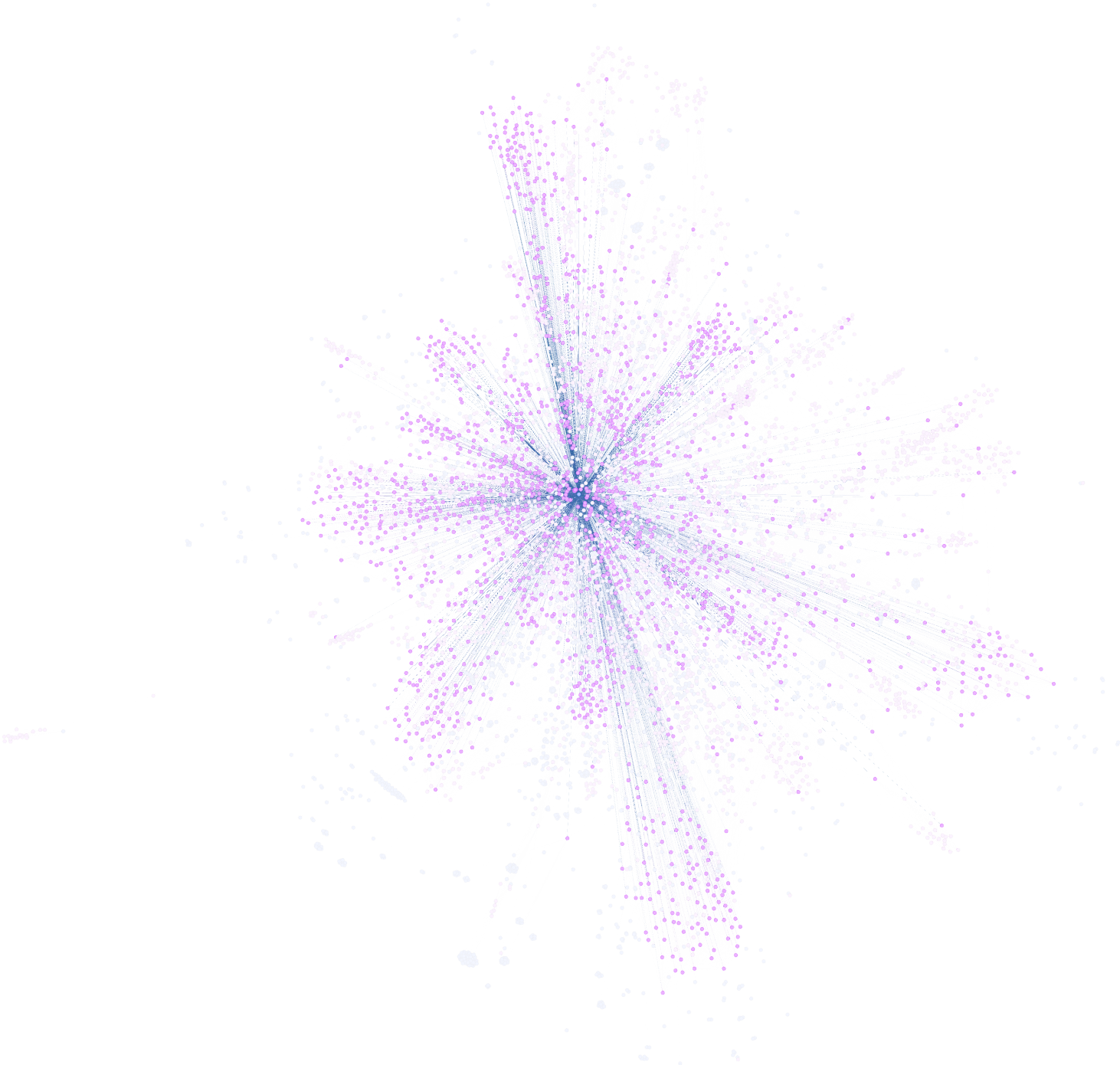


Figure 4.4: Streamers and games in graph network with most popular game *Just Chatting* highlighted. Pink nodes are streamers; blue nodes are games.

Some of the data used in the analysis.

Table 4.1: top 10 streamers by betweenness

name	**views_avg**	**tfidf_score**	**betweenness**	**clusterID**
esl_csgo	15028	0.015643275	0.06843872629380872	15
shroud	17814	0.119327649	0.04027987787945484	16
xqc	57999	0.091482006	0.040069984361607754	16
otplol_	9533	0.001890359	0.034243000548030755	23
foolish_gamers	6988	0.158491572	0.03248278489686211	20
krl_stream	1301	0.003508772	0.032289837468070376	15
summit1g	17310	0.100935789	0.03152296828251939	16
kaicenat	57645	0.065497628	0.02828144371874025	31
place	2295	0.004150845	0.02795975287638164	17
aspaszin	10992	0.001912046	0.0260724840101937	31

Table 4.2: top 10 streamers by tf-idf

name	**views_avg**	**tfidf_score**	**betweenness**	**clusterID**
ai_sponge	6768	1	0.001080759702814037	16
skzlatamproject	5979	1	0.0	12
pronewstv	1584	1	0.0	24
juliapitzer	2474	1	0.0	18
joltzdude139	1224	1	0.0	21
himetokki	1597	1	0.0	25
oreitroll	2155	1	0.0	26
superzomgbbq	1646	1	0.0	28
ruwarface	1941	1	0.0	17
wolviehd	1259	1	0.0	20

Table 4.3: top 10 streamers by avg. views

name	**views_avg**	**tfidf_score**	**betweenness**	**clusterID**
kingsleague	304985	0.007575758	2,54E-12	20
evelone192	91097	0.002029262	5,19E-11	17
thegrefg	83245	0.005539516	0.00142583721354589	20
austinshow	80671	0.000549753	0.0	16
auronplay	80200	0.073886222	7,40E-11	20
ibai	78567	0.004991274	0.0032769028420825217	20
paulinholokobr	78441	0.067664913	1,33E-12	31
elspreen	78124	0.035979881	0.003024240390287704	20
adinross	73424	0.00149737	6,47E-10	31
asmongold	66555	0.009708738	0.0	29

Table 4.4: community scores

Community ID	**Streamer Count**	**Median View**	**Avg. Followers**	**RW inflow sum**	**RW retention**	**M20 inflow sum**	**M20 retention**	**M50 inflow sum**	**M50 retention**
19	112	2277.0	45799.0	0.64	0.63	0.67	0.62	0.39	0.32
1	192	2088.0	148355.0	1.01	0.93	1.11	0.67	1.15	0.39
5	509	2084.0	705328.0	1.46	0.98	1.57	0.84	2.15	0.67
13	102	2082.0	240747.0	0.97	0.96	0.47	0.44	0.19	0.14
7	168	2068.0	805469.0	1.01	0.71	1.16	0.19	1.02	0.06
15	187	1938.0	211006.0	1.05	0.98	1.22	0.86	1.44	0.7
18	251	1933.0	284237.0	1.02	0.98	1.12	0.76	1.2	0.52
14	337	1910.0	207172.0	1.08	0.8	1.07	0.18	0.94	0.06
8	176	1882.0	990893.0	1.11	0.93	1.27	0.56	1.26	0.26
16	87	1858.0	149722.0	1.0	0.98	1.07	0.95	1.15	0.88
17	277	1851.0	496928.0	1.12	0.73	0.65	0.06	0.56	0.03
9	379	1815.0	356188.0	1.93	0.99	1.65	0.76	2.17	0.53
3	69	1810.0	315014.0	0.9	0.77	0.42	0.12	0.32	0.02
22	110	1795.0	376717.0	1.02	0.99	1.03	0.98	1.06	0.96
2	300	1792.0	314033.0	0.82	0.7	1.04	0.26	0.96	0.07
10	412	1791.0	250061.0	1.23	0.98	2.29	0.85	2.99	0.7
6	96	1764.0	383892.0	0.83	0.71	0.4	0.12	0.33	0.02
21	90	1758.0	377447.0	1.0	0.99	0.67	0.58	0.43	0.27

Note that above community IDs do not directly correspond to the cluster IDs used in section 4.3 since the modularity (Louvain method) was computed twice independently resulting in almost identical clusters but with different randomly allocated cluster IDs.

Cluster Characterizations by Games and Tags

c17 = {0.16184982414307092: 'Just Chatting', 0.07104980881651908: 'Dota 2', 0.06135680661901802: 'Virtual Casino', 0.03240491894583128: 'Counter-Strike: Global Offensive', 0.021604489551049322: 'Hogwarts Legacy', 0.016897821248236022: 'Slots', 0.014941295235719475: 'ASMR', 0.014931970447338754: 'Minecraft', 0.014427368442149101: 'PUBG: BATTLEGROUNDS', 0.011848564126954198: 'Sons of the Forest', 0.010310439540109347: 'Grand Theft Auto V', 0.01030958280548526: 'Fortnite', 0.010238217563358075: 'World of Tanks', 0.009796366944639392: 'Call of Duty: Warzone', 0.008721456125603126: 'Dominance', 0.008260247418320826: 'Pools, Hot Tubs, and Beaches', 0.008245662176581318: 'Dead Space', 0.007740407320760437: 'Atomic Heart', 0.007235459382533799: 'StarCraft II', 0.007226694058190095: 'VALORANT'}
c17_tag = {0.21220629353394424: 'Русский', 0.13887170910642516: 'English', 0.044864280503524694: 'DropsВключены', 0.019820727342406017: 'Español', 0.015185710135022968: 'Esports', 0.013185546158962825: 'DropsEnabled', 0.012450990710130701: 'Suomi', 0.011143800084494589: 'русский', 0.008529418832924107: 'dota2', 0.007938962652628386: 'IRL', 0.007931533293146073: 'ASMR', 0.007875823520062254: 'Киберспорт', 0.006568632894320809: 'Казино', 0.005957318867296057: 'woman', 0.005951058821120193: 'english', 0.005931973937364487: '한국어', 0.005915037581463206: 'casino', 0.005298805010326711: 'Anime', 0.005292851340660118: 'anime', 0.005290346736481611: 'Deutsch'}

c16 = {0.055561043865724493: 'Just Chatting', 0.048223983685784386: 'Escape from Tarkov', 0.018767871337179807: 'Dead Space', 0.018421696590123576: 'Sons of the Forest', 0.014999116678382492: 'Dark and Darker', 0.011782908961034363: 'Destiny 2', 0.010397287479144462: 'Hogwarts Legacy', 0.0097059058802004: 'Elden Ring', 0.009326066729689: 'Minecraft', 0.008825635603263259: 'Wo Long: Fallen Dynasty', 0.008550385889736594: 'HITMAN World of Assassination', 0.008518703045522583: 'Resident Evil 4', 0.008255687870280057: 'Hunt: Showdown', 0.008247878987098606: 'Dead by Daylight', 0.006740115671521659: 'Special Events', 0.006488996584920256: 'DayZ', 0.005705921587528037: 'League of Legends', 0.005607113965282019: 'Art', 0.005589357299121663: 'Forspoken', 0.005292121713814449: 'Apex Legends'}
c16_tag = {0.24390420185271083: 'English', 0.09958658282327033: 'DropsEnabled', 0.01642122696447753: 'Vtuber', 0.016272982307318284: 'Anime', 0.013833422983616683: 'vtuber', 0.011406384939806015: 'ADHD', 0.010841347731314314: 'PvP', 0.00883621405739794: 'english', 0.008633442668260593: 'anime', 0.00816038137446962: 'drops', 0.0077813022080205855: 'DropsAktiviert', 0.007053358711587847: 'AMA', 0.007004134304680039: 'VTuber', 0.006481184846678445: 'Speedrun', 0.0062980820472351325: 'Educational', 0.006179657371580332: 'Chatty', 0.005972882631799138: 'Drops', 0.005845453687984726: 'adhd', 0.005791808091904674: 'Chill', 0.005732762701699304: 'pvp'}

c31 = {0.09005904498718328: 'Just Chatting', 0.07819998397549759: 'VALORANT', 0.06286368557433847: 'Grand Theft Auto V', 0.04011532678362641: 'Apex Legends', 0.030001103440092183: 'Overwatch 2', 0.029974164618592366: 'League of Legends', 0.027421244224436685: 'FIFA 23', 0.019343293859518426: 'Hogwarts Legacy', 0.01854604985097484: 'Sons of the Forest', 0.01681100671877551: 'Escape from Tarkov', 0.01384655402748277: 'Counter-Strike: Global Offensive', 0.013771043780848993: 'Minecraft', 0.01307515966557087: 'Virtual Casino', 0.012151192885389616: 'Teamfight Tactics', 0.011927834418033752: 'Fortnite', 0.00999937566645053: 'Dark and Darker', 0.008943503337196393: 'Rust', 0.007721358671207551 6: 'Red Dead Redemption 2', 0.006982444734841928: 'Super Smash Bros. Ultimate', 0.006475264742919273: 'Among Us'}
c31_tag = {0.15764020357539235: 'Português', 0.1473606988785523: 'English', 0.04666173613775747: 'DropsAtivados', 0.0439646358124026: 'Italiano', 0.02617570757259319: 'ไทย', 0.02599343833518329: 'DropsEnabled', 0.01371223666105065: 'Roleplay', 0.012918934047514303: 'Esports', 0.012505741427498928: 'DropAbilitati', 0.008354158040264738: 'Arabic', 0.007333695578533251: 'rp', 0.007170396203449907: 'gaming', 0.007164954049809335: 'Magyar', 0.007151271739399683: 'gtarp', 0.006808933170135947: 'roleplay', 0.0065600899086568675: 'GTARP', 0.006530345951686326: 'drops', 0.006401094675138571: 'fps', 0.006007373235492907: 'Español', 0.005975750053623893: 'FIFA'}

c20 = {0.13953854004209565: 'Just Chatting', 0.07114268397925745: 'Minecraft', 0.04100701667320792: 'Counter-Strike: Global Offensive', 0.03698972394164368: 'League of Legends', 0.03590969608172438: 'VALORANT', 0.0237206414198306: 'Hogwarts Legacy', 0.023409937826662498: 'Grand Theft Auto V', 0.022412960757692325: 'FIFA 23', 0.02029091540346842: 'Sons of the Forest', 0.01964491716456854: 'Sports', 0.014965927740301465: 'Fortnite', 0.013408746265256228: 'Call of Duty: Warzone', 0.012190825423885202: 'Travel & Outdoors', 0.010631406323894877: 'Fall Guys', 0.00939585204706454: 'Roblox', 0.009078110072745603: 'Among Us', 0.008357840851425413: 'Escape from Tarkov', 0.007829467818950308: 'Special Events', 0.00753183228934697: 'ROBLOX', 0.007438903438084376: 'Slots'}
c20_tag = {0.3542615217016583: 'Español', 0.08978483894805271: 'DropsActivados', 0.046329608007399554: 'English', 0.015151657727845465: 'español', 0.01119684454906328: 'minecraft', 0.010233121778962618: 'IRL', 0.009584119455066093: 'squidcraftgames2', 0.007993394234289016: 'argentina', 0.007993394222132031: 'Argentina', 0.007198031623914056: 'futbol', 0.007198031623362393: 'Latino', 0.006572206442281177: 'Anime', 0.006402669013536932: 'Futbol', 0.006402669013527058: 'fútbol', 0.006384124464794922: 'valorant', 0.006084020461285501: 'PequeñaGranComunidad', 0.0056073064031459795: 'colombia', 0.004811943792785715: 'charlando', 0.004778014843081478: 'Minecraft', 0.0047728785754684: 'fifa'}

c13 = {0.09005904498718328: 'Just Chatting', 0.07819998397549759: 'VALORANT', 0.06286368557433847: 'Grand Theft Auto V', 0.04011532678362641: 'Apex Legends', 0.030001103440092183: 'Overwatch 2', 0.029974164618592366: 'League of Legends', 0.027421244224436685: 'FIFA 23', 0.019343293859518426: 'Hogwarts Legacy', 0.01854604985097484: 'Sons of the Forest', 0.01681100671877551: 'Escape from Tarkov', 0.01384655402748277: 'Counter-Strike: Global Offensive', 0.013771043780848993: 'Minecraft', 0.01307515966557087: 'Virtual Casino', 0.012151192885389616: 'Teamfight Tactics', 0.011927834418033752: 'Fortnite', 0.00999937566645053: 'Dark and Darker', 0.008943503337196393: 'Rust', 0.007721358671207551 6: 'Red Dead Redemption 2', 0.006982444734841928: 'Super Smash Bros. Ultimate', 0.006475264742919273: 'Among Us'}
c13_tag = {0.2642460363941913: 'English', 0.09142855587628333: '日本語', 0.06877776416157674: 'DropsEnabled', 0.019025405571119258: 'Drops有効', 0.014360615849924993: 'Esports', 0.01127889255096795: 'Competitive', 0.009951740071290863: 'Español', 0.00856927352965379: 'ADHD', 0.0072066449468599705: 'Music', 0.007188645063364892: 'music', 0.007155718809880 98: 'Musician', 0.006561883223999044: 'musician', 0.006159659570051873: 'Français', 0.006052147364425947: 'Chill', 0.005632719216246438: 'singing', 0.005584745244759438: 'Singing', 0.005555049791429514: 'FPS', 0.005411455188287142: 'DropsActivados', 0.005374212049345582: 'Guitar', 0.004987664313584203: 'Tournament'}

c14 = {0.0918252214361464: 'Just Chatting', 0.03151536575647246: 'Fortnite', 0.028761230379663966: 'Hogwarts Legacy', 0.023954232920893742: 'Sons of the Forest', 0.0225910590839237: 'Minecraft', 0.022151556602151064: 'Rocket League', 0.02096547212133026: 'League of Legends', 0.020585249922573583: 'VALORANT', 0.01944116020650964: 'Grand Theft Auto V', 0.01816131894506856: 'Special Events', 0.017327776127692995: 'Call of Duty: Warzone', 0.013639002825168327: 'Call of Duty: Modern Warfare II', 0.011368988478784505: 'FIFA 23', 0.009424650217153639: 'Dead Space', 0.009330507574849718: 'Among Us', 0.008815222352436614: 'Sea of Thieves', 0.008744798093640609: 'Sports', 0.007917941315638211: 'Talk Shows & Podcasts', 0.007590056955733206 5: 'Fall Guys', 0.007112711598490865: 'Forspoken'}

c14_tag = {0.22144998340621633: 'Deutsch', 0.12461800835179114: 'English', 0.05673784797022443: 'DropsAktiviert', 0.04133110753009586: 'DropsEnabled', 0.014949878106164983: 'Esports', 0.014694069044359343: 'Fortnite', 0.014667463398809455: 'deutsch', 0.013811353608329585: 'Čeština', 0.008081600177288045: 'FamilyFriendly', 0.007819012741566125: 'Competitive', 0.007445168934785203: 'Español', 0.006796622618744305: 'IRL', 0.00646627841492927: 'PovolenéDrops', 0.005671952819603297: 'english', 0.005555642930100378: 'Nederlands', 0.005408729496404717: 'Ranked', 0.005313523513012044: 'Drops', 0.005223769455950654: 'german', 0.005140878849590727: 'Gaming', 0.0048628028858002294: 'Variety'}

c23 = {0.0918252214361464: 'Just Chatting', 0.03151536575647246: 'Fortnite', 0.028761230379663966: 'Hogwarts Legacy', 0.023954232920893742: 'Sons of the Forest', 0.0225910590839237: 'Minecraft', 0.022151556602151064: 'Rocket League', 0.02096547212133026: 'League of Legends', 0.020585249922573583: 'VALORANT', 0.01944116020650964: 'Grand Theft Auto V', 0.01816131894506856: 'Special Events', 0.017327776127692995: 'Call of Duty: Warzone', 0.013639002825168327: 'Call of Duty: Modern Warfare II', 0.011368988478784505: 'FIFA 23', 0.009424650217153639: 'Dead Space', 0.009330507574849718: 'Among Us', 0.008815222352436614: 'Sea of Thieves', 0.008744798093640609: 'Sports', 0.007917941315638211: 'Talk Shows & Podcasts', 0.0075900569557332065: 'Fall Guys', 0.007112711598490865: 'Forspoken'}
c23_tag = {0.42540095463201555: 'Français', 0.05903266846149706: 'English', 0.014527715383655581: 'Esports', 0.014420700967356993: 'DropsEnabled', 0.012845956218836104: 'Español', 0.01147841682392511: 'chill', 0.011295405456832015: 'Português', 0.009610659273904555: 'français', 0.008089612322681211: 'react', 0.008022120550995337: 'discussion', 0.008022120550977244: 'RocketLeague', 0.008022120550945077: 'DropsActivés', 0.006657707793537817: 'fun', 0.006449095488464216: 'Football', 0.006445099728047491: 'gtarp', 0.006433581828278814: 'News', 0.006433581828120482: 'Esport', 0.005513215809778776: 'gaming', 0.004872563389243353: 'drops', 0.004848744887170232: 'DropsAtivados'}

c22 = {0.09277370183547412: 'Just Chatting', 0.04856904960789501: 'League of Legends', 0.03578800862537164: 'Genshin Impact', 0.029288261541399064: 'Rust', 0.028246512196620693: 'Minecraft', 0.02299414292966005: 'Escape from Tarkov', 0.018569238435061447: 'Sons of the Forest', 0.0181902493111963: 'VALORANT', 0.015027304465351426: 'KartRider: Drift', 0.013136674875288535: 'Dead Space', 0.013105783512426654: 'Games + Demos', 0.013091947073705318: 'Project Zomboid', 0.012784030317532816: 'Lost Ark', 0.012446114053846165: 'Hogwarts Legacy', 0.012087228950215205: 'Dark and Darker', 0.011687996789296095: 'Teamfight Tactics', 0.01128817746873915: 'Wo Long: Fallen Dynasty', 0.010515925677974826: 'VRChat', 0.009308009975028363: 'FIFA Online 4', 0.009016715398327718: 'Overwatch 2'}
c22_tag = {0.17449924268823563: '한국어', 0.11863510076041013: 'English', 0.081032689315414: 'Polski', 0.0774017920025181: '드롭활성화됨', 0.035887362019902364: 'DropsEnabled', 0.033106782335525727: 'DropyWłączone', 0.019350533191653387: 'Español', 0.017558344705926782: 'Esports', 0.011765023269885813: 'DropsActivados', 0.011615428456268112: 'Vtuber', 0.011389272433623478: '브이튜버', 0.008662743400008137: 'Anime', 0.006762143934876506: 'vtuber', 0.006723224675410879: 'ASMR', 0.006663433249545338: '버튜버', 0.006605364916312567: 'leagueoflegends', 0.005718265412725923: 'Gacha', 0.005470019535806108: 'ADHD', 0.004831918357955359: 'anime', 0.004773097575911683: 'rust'}