KU LEUVEN

Advanced Analytics in Business [D0S07a]

Lab Report

Prof. Dr. Seppe vanden Broucke

Academic year: 2022-2023

Group 15:
Michal Jan Chmura - r0960234
Lunkyadi Sucipto - r0916778
Aliz Marossy - r0960753
Naichuan Zhang - r0913147
Christian Daniel Sepulveda Arzuza - r0821283
Maximilien Daoût - r0886713

Contents

1 Assignment 1: Predictive Modeling using Tabular Data

In this assignment we constructed a predictive model to predict the "price per night" of AirBnB properties in Belgium. In our report, we will discuss the steps in the process of creating and using the model, including pre-processing, model creation and the training of the model. Then, we will introduce the results and discuss the issues that may become apparent.

1.1 Pre-processing

We were given a "dirty" tabular dataset, with which we are to regress the asking (rent) price of an AirBnB. The filthiness here involves typos (fixed manually), systematically missing values, and a non-ideal data format. In the data set, there are numerical, categorical and text features. So we work on both structured and unstructured data. The textual data are free-response columns such as `property_desc` and `property_summary`, which describe the room features.
We decided to work with ten datasets in total, with eight of them being all possible combinations of:

- PCA and not PCA
- `property_type` & `property_room_type` frequency encoded or one-hot encoded
- Outliers removed and not removed

The eight mentioned do not include textual data. We created an additional two datasets, whether `property_type` and `property_room_type` are frequency encoded or one-hot encoded, joined it with the textual data, and PCA them with 70% of the total variance.

1.1.1 Missing Values

We figured that all AirBnBs have at least one bathroom and no bedrooms (which is indeed very common in the dataset), so missing values for `property_bedrooms` and `property_bathrooms` was set to zero and one, respectively. We think, a "bedroom" is only counted as one if there is a separation, say between a living room and a bedroom (so, studios would count as no bedroom). Similarly, we found that a ratio of two people per bed was almost universally true, so we set missing `property_bedrooms` to half of `property_max_guests`, rounded up.

The values in the review columns are systematically missing, i.e., if a property has never been reviewed, then all of its columns are set to *NaN*. Strictly speaking, this isn't a missing value, all information (that there is none) is properly recorded, and we have set `reviews_num` to zero.

Column `property_zipcode` contained missing values, but `property_lat` and `property_lon` don't, we think that these were mandatory fields when one puts up their property to rent.

As a fix, we used `GeoPy`[1] to find the zipcode based on `property_lat` and `property_lon`.

For the missing numerical values (all of `reviews_xxx` except `reviews_num` and `host_response _rate`), we decided to impute them with the median. As for the missing text value, we replace the *NaN* with a space string " ". *NaNs* are not informative, but cannot be analyzed as text.

1.1.2 Feature Engineering

Since Scikit-learn can only work with numerical data, the two most obvious things to do is something with the categorical and ordinal columns. There is not much to say about the latter, we simply created an integral order for `host_response_time` and `booking_cancel_ policy`. In the former, for the `property_type` and `property_room_type` columns, we worked with either all frequency-encoded or all one-hundred-encoded data, creating two versions of the data. The `host_id` column is always frequency encoded.

The column `host_location` contains entries such as "*Brussels, Brussels, Belgium*". We turned these into a column indicating if the host lives in the same city (nearby) as the listed property. However we only processed entries for properties in Brussels and Antwerp, but these cover 6300/6495 of the cases.

Next are what we call "checklist-based" transformations. Columns `host_ verified`, `property_ amenities`, and `extra` have entries of this format:

- `email, phone, reviews`
- `email, phone, facebook reviews, jumio`
- ...

We think that these are aggregated from a list of boolean values, sort of like a checklist. For each item in the list, we give it one point. Thus, the `host_verified` entry for the first bullet point would be 3, the second bullet point would have 4, etc.

There are three time-related columns, `property_scraped_at`, `host_since` and `property_ last_updated`. We decided to throw out `property_scraped_at` since it contained only three unique values, 2017-05-08, 2017-05-12, and 2017-05-09, and the fact that this piece of information has nothing to do whatsoever with the target variable.
However, we deemed that there is at least one piece of information based on `property_ scraped_at`, namely we know that the data is "fresh" around 2017-05-12, the last day of the scraping. We used this date as "today" when we processed `host_since` and `property_last_updated`. For both columns, we figured out how long ago (in months) the host started/ the property was last updated.

Next, we extract features from textual data. For `property_type`, `property_room_type`, `property_bed_type`, they only have several categories, so we apply one-hot encoder on them. The `property_last_updated` contains periods in different time units, such as 5 days and 6 months. We transform all the units into days. For example, we change 6

[1] https://geopy.readthedocs.io/en/stable/

months into 180 days. All the text data are not in one single language, which can bring us much trouble to extract information from text. We translate all the text into English using Google API. After doing this work, we process textual data. We transform all the text into lowercase, remove the emojis in the text, remove the stop words (such as "a" and "on"), correct the spelling, tokenize the text, find the stem words and lemmatize the text.

With the pre-processed textual data, we utilise the TF-IDF method to extract features from the text. TF (Term Frequency) of a term or word is the number of times the term appears in a document compared to the total number of words in the document. IDF (Inverse Document Frequency) of a term reflects the proportion of documents in the corpus that contain the term. The resulting TF-IDF score reflects the importance of a term for a document in the corpus. We trained a BoW featurizer to convert the text into a format that SKLearn can work with.

Finally, we combine all the features together and continue to do the feature selection.

1.1.3 Feature Selection

We found that the columns `host_nr_listings` and `host_nr_listings_total` are the exact same, so one has to go. Furthermore, we think that the information here is already captured[2] in `host_id_freq` anyway when we frequency encoded the `host_id` column, so we decided to drop them.
When we performed the PCA, we took 80% of the total variance for the text-free data, and 70% for the two datasets containing text. For the (one-hot)version of the text-free data, we started with 42 feature columns and ended up with 22. For the (freq) version, we started with 36 and ended up with 18.

1.1.4 Outlier Detection

We used Isolation Forest to perform an outlier analysis and throw out from the training data the outliers. We use SciKit-Learn's[3] implementation here. Isolation Forest classifies an observation as either an outlier or not.

From testing, we found that IsolationForest runs very quickly, and thus we were able to comfortably set the number of trees to 10000. We left the `max_samples` to the default of 256, since it is recommended to use a small amount of observations in the original article to prevent masking (drawing too many outliers that it results in the outliers being masked). We also decided to keep the default of using all features for every tree.

There is an argument of whether or not we should set `bootstrap=True`. Upon testing on the previous paragraph's settings, we found that out of 6495 observations, `bootstrap` only causes a difference of classification on 9 observations. Furthermore, the classification is based on `score` $\leq 0 \implies$ `outlier`, and the borderline cases observations with `abs(score)` ≈ 0.002. We thus decided to set `bootstrap=True`.

[2] Idea is that if they have multiple properties, then their `host_id` will appear multiple times and their `host_id_freq` will be high. However upon writing this report we just realized that the total number of listings may include property outside of Brussels/Antwerp, so this may have been a mistake.

[3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

Unfortunately, after detecting the outliers, it is not easy to tell what makes an observation an outlier. That is indeed a weakness of Isolation Forest as a technique, we only get a pure numerical score. We can't try checking via Mahalanobis distances what causes an outlier, since our data violates the elliptically distributed assumption (e.g. the frequency encoded features).

1.2 Model creation and training

After some trials, we decided to use the combination of XGBRegressor from the XGBoost library, and RandomForestRegressor from the sklearn library. One might ask why use two models which have roughly the same basic principle - we wanted to try using different models from different libraries for the purpose of practice, and accidentally discovered, that combining the two "champion" models (more about choosing the hyperparameters later) gives slightly better result. For the weights, we decided to go with a 50/50 proportion, as the ones derived from the models' prediction scores were really close to that.

1.2.1 Hyperparameter tuning

For the purpose of assessing models' scores, data was split in proportion 80 : 20. After that, hyperparameter tuning was conducted, with the use of RandomizedSearchCV function from sklearn library, with cross-validation set on 5. We decided against using GridSearchCV, because of lack of resources. The parameters searched are included in the list below:

parameters	tuning set
n_estimators	[**100**, 200, 300]
max_depth	[**3**, 5, 7, 8]
min_child_weight	[3, **5**]
gamma	[**0.3**, 0.5]
eta	[0.03, 0.05, **0.07**, 0.1]
subsample	[**0.8**, 0.6]
colsample_bytree	[**0.1**, 0.2]
lambda	[1, 1.5, 2, **3**]
alpha	[0, 0.1, **0.5**, 1]
booster	[**'dart'**]

Table 1: Parameters tuning set of XGBoost regressor (best parameters are in bold)

parameters	tuning set
n_estimators	[100, 200, 300, **400**, 500]
max_depth	[**5**, 6, 7, 10, 12, None]
max_features	[1.0, **'sqrt'**]
min_samples_leaf	[1, 2, **4**]
min_samples_split	[2, **5**, 10]
bootstrap	[**True**, False]

Table 2: Parameters tuning set of Random Forest regressor (best parameters are in bold)

1.3 Limitations and Conclusion

We ended up on a 10th place in the final leaderboard. It is a slight improvement over the public leaderboard, but not much. Our MAE final score was much better though, so it hints that the model was not so bad compared to the others. We could have squeezed more performance from the text data, as we only used basic techniques and translation of all descriptions to English. Final score was also more than 20% worse than the public score, which strongly hints overfitting. We could have ensembled other approaches. Even by adding small weight to linear regression model which was briefly considered (all the files and version changes can be found on our project's github page linked in the references). Perhaps there is a simple but efficient method that we missed due to limited time (like the one that got the 3rd, 4th and 5th team identical scores in all 4 categories, one of them in just 2 attempts)? To summarize, we should have gone with the Pareto Principle and spend 80% of time and manpower on data wrangling, not on finding better and better hyperparameters.

2 Assignment 2: Deep Learning on Images

The objective of the second assignment is to develop a deep learning model capable of predicting the price of services offered by a restaurant based on the images of it. In our report, we will discuss the steps in the process of creating and using the model. Then, we will introduce the results and discuss the issues that may become apparent.

2.1 Model Training

In the second assignment, we were given three potential goals to choose from. As a team we decided to do an image classification task where given pictures of a dish (or the restaurant itself), we have to classify the price range:

1. On a budget (44% of trainset)
2. Moderate spend (29%)
3. Special occasion (15%)
4. Spare no expense (10%)

Do note that there are a (very) small number of restaurants that are unlabeled, and we decided to throw away these from the trainset.
To achieve this end, we built two models, one utilizing Keras and another with PyTorch. See Figure 1 for some examples of the images.

(a) Image 259484 (b) Image 439331

Figure 1: Images part of the train set

2.1.1 Tensorflow's Keras

The link to access the code for this part is available in the references.

Due to computational power and time constraints, we based the model training on a version of the data with fewer images than the original folder: stratified images (1700 images). It will allow us to build our model without having to load the 117000 images of the initial folder and to save a significant amount of time in the training process. Then

we reduce the metadata to the images of this stratified folder.

After importing the images and attaching them to the corresponding meta data, we split it into training and validation sets.

Model Building We start implementing the model using Keras. Here we use a pre-trained VGG 16 model, which contains 13 convolution layers and 3 fully connected layers, the classification layers. It is a very popular convolutional neural network for image classification, to which we add 4 dense layers (custom head). The first 3 layers have a ReLU activation function (Rectified Linear Unit). This activation function is often used in deep neural networks because they allow nonlinearity to be introduced into the outputs of each layer. Indeed, images can have complex (non-linear) relationships between pixels and class labels. It also appears that this function are used in practice because they are simple and have been shown to perform well empirically for image classification problems (He et al., 2015). Finally, the last layer is activated using a softmax activation function that allows the outputs of the dense layer to be normalized into a probability distribution over the four classes.

1. Pre-trained VGG16
   (a) 13 Convolutional layers
   (b) 3 Fully connected layers
2. Custom head
   (a) Dense layer: 128 neurons (ReLU activation)
   (b) Dense layer: 64 neurons (ReLU activation)
   (c) Dense layer: 32 neurons (ReLU activation)
   (d) Dense layer (output): 4 neurons (Softmax activation)

We then use the Adam optimizer to minimize our loss function, because of its learning speed. In this paper we use the loss function "categorical_crossentropy" (Eq. 1) which is a particularly suitable loss function for multiclass classification problem. Indeed, It measures the divergence between the probability distribution predicted by the model and the actual probability distribution of the class labels.

$$CE = -\sum_{i}^{C} t_i \log(s_i) \tag{1}$$

Where t_i and s_i are the ground truth (for instance, [1, 0, 0, 0] if the true class is *On a budget*) and model scores (for the same example some probability estimates for each class, something like [0.83, 0.11, 0.04, 0.02]) for each class i in C.

Image Augmentation The next step in this process is image augmentation. We use this technique to improve the quality of the data and thus the performance of the model. We could not use the entire training set, but we still want robust models, so we expanded the images we used in training the model. This technique consists in generating new data by zooming, rotating, shifting or flipping (Figure. 2). In general, it allows diversification of the data provided to the model. It can particularly help in the context of our model,

since all the data looks “expensive”. This step is expected to limit the overfitting that occurs due to the nature of the images available to us.

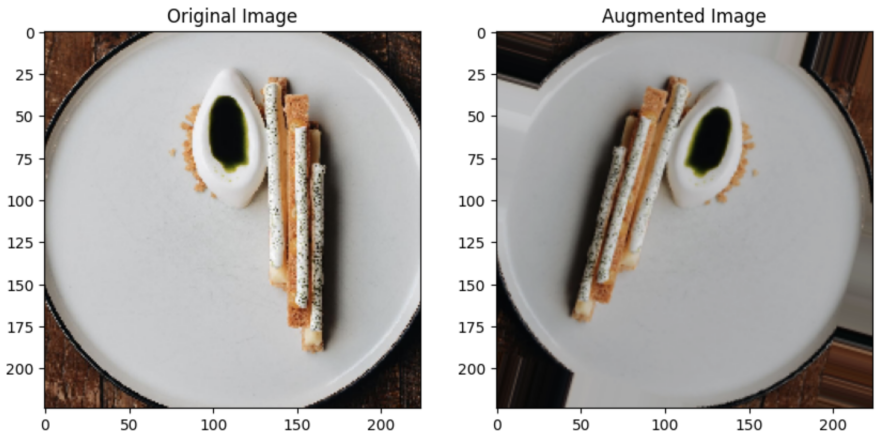


Figure 2: Image augmentation example

Results Figure 3 shows the results for accuracy and categorical cross entropy loss over epochs. It is noticeable that the loss already starts to increase after the first epoch: we suspect that this is due to us prematurely trying to augment images before having a fully-trained network. In any case, we set our final model after only one epoch.

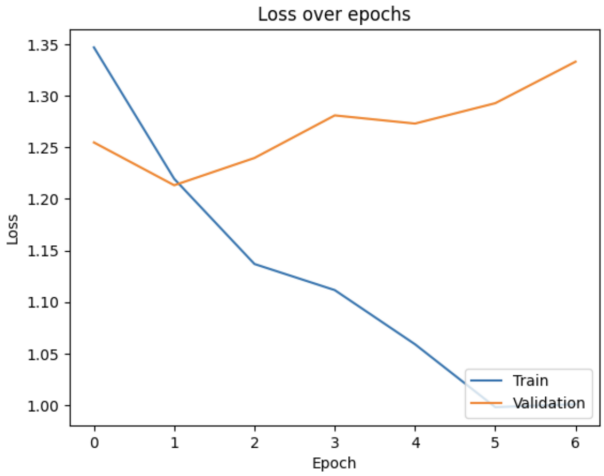


Figure 3: Loss over epochs

Figure 4 seems to indicate that our model only learnt how to classify an image as "on a budget" or not, and even rather poorly at that.
To begin with, our labels are ordinal, so we expect that if the model learnt something, it would mis-classify in a sensible way. Ideally we would see that even if mis-classified, most of the misclassifications would go to "a moderate spend", then less mis-classifications into "special occasion", and even less on "spare no expense". However the complete opposite is reported here.
Furthermore, the "spare no expense" observations are more often classified as "on a budget" than itself, further suggesting that the model did not learn about any sort of ordering here. Curiously, 73% of the data are labeled "a moderate spend" or "special occasion",

but the model hardly predicts these two. Rather, it feels as if the model tried to learn how to predict the most minor class.

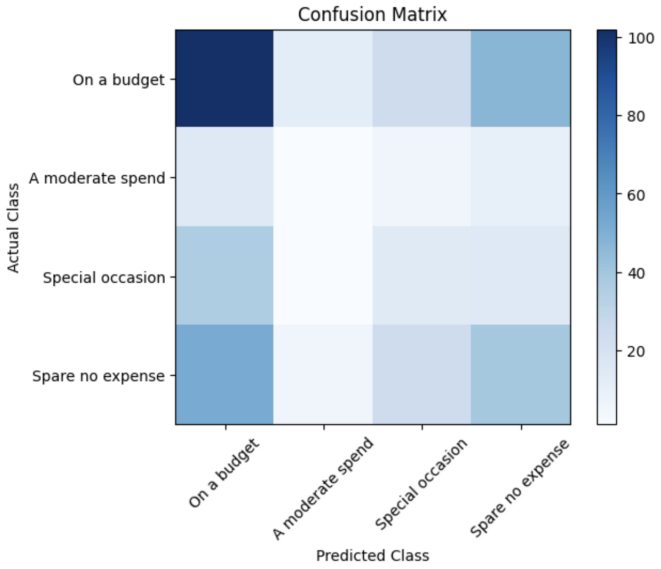


Figure 4: Confusion matrix of the classification task

Figure 5 illustrates an example of this misclassification problem.

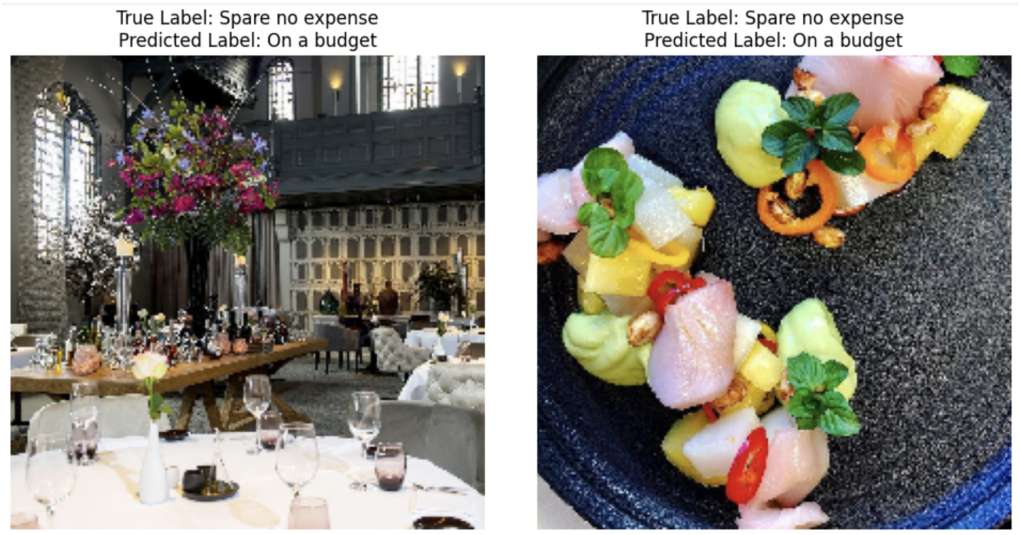


Figure 5: Image illustration of important misclassification

The results of implementing the network with Keras do not meet our expectations when we analyze the metrics. For this reason, we decided to implement a second neural network with a different library, hoping to obtain better results.

2.1.2 EfficientNet_V2_S for transfer learning

The pretrained architecture of EfficientNet_V2_S is tested in a transfer learning scheme using PyTorch, the architecture of the network can be seen in Figure 6. It is mostly formed by different blocks that use convolutions and residual connections, MBConv and FusedMBConv, traditional convolutional layers, Conv2d, layers with the sigmoid linear unit function (SiLU), normalization steps, BatchNorm2d, and a linear layer for the classification. The most notable aspects of this architecture are the residual connections, as this solves the problem of the vanishing gradient and allows to effectively train such a large network. The non-traditional convolution blocks, mostly take advantage of linear inverted bottlenecks to abstract features, depending on the layers this is done at different levels of abstraction.

This transfer learning scheme is used to take advantage of the learning that the network has already done. This saves a tremendous amount of time and resources, as low-level features, like edges and very simple shapes, do not need to be learned from scratch. Furthermore, when the task that the network was originally trained for has a big similarity with the current one, even high level features are useful after some fine-tuning. In this case, EfficientNet_V2_S was originally trained to classify images from the ImageNet data set into 1000 classes. This is a rather general task compared to classifying restaurants in price categories, so it is expected that only the low level features learned by the network would be useful for this application. Given this, all network's blocks, sub-groups of layers, are fine-tuned, with the exception of the first three. This is done like this, as the first ones are the ones that detect the low-level features, while the other six blocks detect the higher level features, with the last one performing the classification, this one is referred to as classifier head.

First, the classifier head of the network is replaced with a new one, now with four outputs, and the other layers' weights are frozen, then the network is trained until the error on a validation set starts to rise. This is called feature-extraction. The next step is the fine-tuning. Here, all the subsequent layers of the network are unfrozen one by one and trained until the error of the validation set starts to rise. This way small adjustments to the network's weights are made to take advantages of the originally learned weights but still be able to tailored the network to our application. This process was done on 10% of the images, as handling more of them is very challenging in google colab, due to the memory constraints, both on GPU memory and RAM. Furthermore, as it takes a relatively very long time to fetch the images in google colab, the images were stored as numpy arrays. In the Pytorch implementation about 10 times more images were used than for the Keras' implementation in order to attempt to obtained better results. Additionally, to keep overfitting under control a relatively high L2 regularization constant had to be used, 0.001, in order for the network not to overfit the data, given the relatively low amount of images available for training a network of this size. Having to use a regularization constant of this magnitude might prevent the network from learning appropriately, but it was necessary otherwise the overfitting was tremendous, reaching up to 85% accuracy on the training set while having 32% accuracy on the validation set. Fine-tuning with regularization was then conducted.

The obtained fine-tuned network has an accuracy of 51% on the test set. It is marginally better than predicting the dominant class which is 48%. This low performance is not

unexpected considering that all restaurants images looked like they belonged to expensive restaurants, so it was a difficult task. Furthermore, it is reasonable that the resulting accuracy is not so high given that the task that the network was originally trained for does not hold a strong similarity to the current one. And crucially, the amount of images that were feasible to use in the Colab environment for training is insufficient to fine-tune a network of this size, and this is the most likely reason behind the model's low performance. Finally, it must be noted that despite all of the above, there was still an improvement gained compared to the network trained using Keras.

2.2 Attempted Inference

We tried to follow `shap.DeepExplainer`[4] to estimate the Shapley values of each pixel of the image. However, multiple issues arose:

- Lack of documentation of `shap`.
- Lack of computing power.
- Internal issues with the `shap` itself.

The two new components to be able to do this analysis is the `DeepExplainer` class and `image_plot` function from `shap`. These do not have any documentation, merely examples of usage. Furthermore, calculating the Shapley values requires a lot of space. Colab had a RAM of 12GB, and still couldn't handle the computation (GPU is not available on the free version). Our strongest laptop had a 8GB RAM. Lastly, when we tried to compute the Shapley values (before crashing), we encountered three warnings:

- `Using a non-full backward hook when the forward contains multiple autograd Nodes is deprecated and will be removed in future versions. This hook will be missing some grad_input. Please use register_full_backward_hook to get the documented behavior.`
- `Warning: unrecognized nn.Module: StochasticDepth.`
- `Warning: unrecognized nn.Module: SiLU.`

All of those warnings relate to the methods being depreciated in the current distribution of the library - we haven't have enough resources to resolve it properly.

[4] https://rb.gy/hyip2

Layer	Param #
EfficientNet (EfficientNet)	--
├─Sequential (features)	--
│ └─Conv2dNormActivation (0)	--
│ │ └─Conv2d (0)	648
│ │ └─BatchNorm2d (1)	48
│ │ └─SiLU (2)	--
│ └─Sequential (1)	--
│ │ └─FusedMBConv (0)	5,232
│ │ └─FusedMBConv (1)	5,232
│ └─Sequential (2)	--
│ │ └─FusedMBConv (0)	25,632
│ │ └─FusedMBConv (1)	92,640
│ │ └─FusedMBConv (2)	92,640
│ │ └─FusedMBConv (3)	92,640
│ └─Sequential (3)	--
│ │ └─FusedMBConv (0)	95,744
│ │ └─FusedMBConv (1)	164,480
│ │ └─FusedMBConv (2)	164,480
│ │ └─FusedMBConv (3)	164,480
│ └─Sequential (4)	--
│ │ └─MBConv (0)	61,200
│ │ └─MBConv (1)	171,296
│ │ └─MBConv (2)	171,296
│ │ └─MBConv (3)	171,296
│ │ └─MBConv (4)	171,296
│ │ └─MBConv (5)	171,296
│ └─Sequential (5)	--
│ │ └─MBConv (0)	281,440
│ │ └─MBConv (1)	397,800
│ │ └─MBConv (2)	397,800
│ │ └─MBConv (3)	397,800
│ │ └─MBConv (4)	397,800
│ │ └─MBConv (5)	397,800
│ │ └─MBConv (6)	397,800
│ │ └─MBConv (7)	397,800
│ │ └─MBConv (8)	397,800
│ └─Sequential (6)	--
│ │ └─MBConv (0)	490,152
│ │ └─MBConv (1)	1,005,120
│ │ └─MBConv (2)	1,005,120
│ │ └─MBConv (3)	1,005,120
│ │ └─MBConv (4)	1,005,120
│ │ └─MBConv (5)	1,005,120
│ │ └─MBConv (6)	1,005,120
│ │ └─MBConv (7)	1,005,120
│ │ └─MBConv (8)	1,005,120
│ │ └─MBConv (9)	1,005,120
│ │ └─MBConv (10)	1,005,120
│ │ └─MBConv (11)	1,005,120
│ │ └─MBConv (12)	1,005,120
│ │ └─MBConv (13)	1,005,120
│ │ └─MBConv (14)	1,005,120
│ └─Conv2dNormActivation (7)	--
│ │ └─Conv2d (0)	327,680
│ │ └─BatchNorm2d (1)	2,560
│ │ └─SiLU (2)	--
├─AdaptiveAvgPool2d (avgpool)	--
├─Sequential (classifier)	--
│ └─Dropout (0)	--
│ └─Linear (1)	5,124

Figure 6: Architecture of EfficientNet_V2_S

2.3 Limitations and Conclusion

By analyzing the results of the two models described in this assignment, we realize that the one using PyTorch seems to perform better (although we can't be sure for the reasons we mention in the limitations about the test set). If we were to keep just one, it would be this one.

Overfitting The results we have obtained indicate the presence of overfitting. As mentioned several times, this is probably due to the fact that not all images differ greatly in the possible classes (cheap, expensive looking restaurants). Another reason is that the images are not uniform: The image set contains photos of dishes, restaurant facades, interiors, etc. With more time and to avoid over-matching, a preliminary screening could have been done to keep only photos of a single type.

Limitations Two elements can be considered as limitations of our approach. First, for technical reasons, we were not able to use the same test set for the two models presented. Namely, the neural network created with Keras required too much time to be build with too large a number of images. Therefore, the construction and evaluation of this network were performed with smaller image samples. Absolute comparisons of the accuracies for the two models should therefore be made with caution.

From this experience it became clear the difficulties of training networks for the application of image classification in a Colab environment. Mainly due to the memory limitations, which in turn limits the amount of images that can be used. This comes as a results of the massive amounts of images that large scale neural networks need in order to be trained. This is true to a lesser extent for fine-tuning, but it still was a limiting factor on our application. Despite this, it is clear that fine-tuning is the preferable option, as it puts in reach deep-learning applications for people that would otherwise not have the resources to train a large scale network from scratch. However, as have been seen in this assignment, this can only achieve great results if the pre-trained network was trained for a similar task or if there are enough images available to fine-tune the network properly.

3 Assignment 3

The following assignment is aimed to construct a model which classifies Steam reviews and predicts whether a review is positive or negative. The solution consists of the following steps:

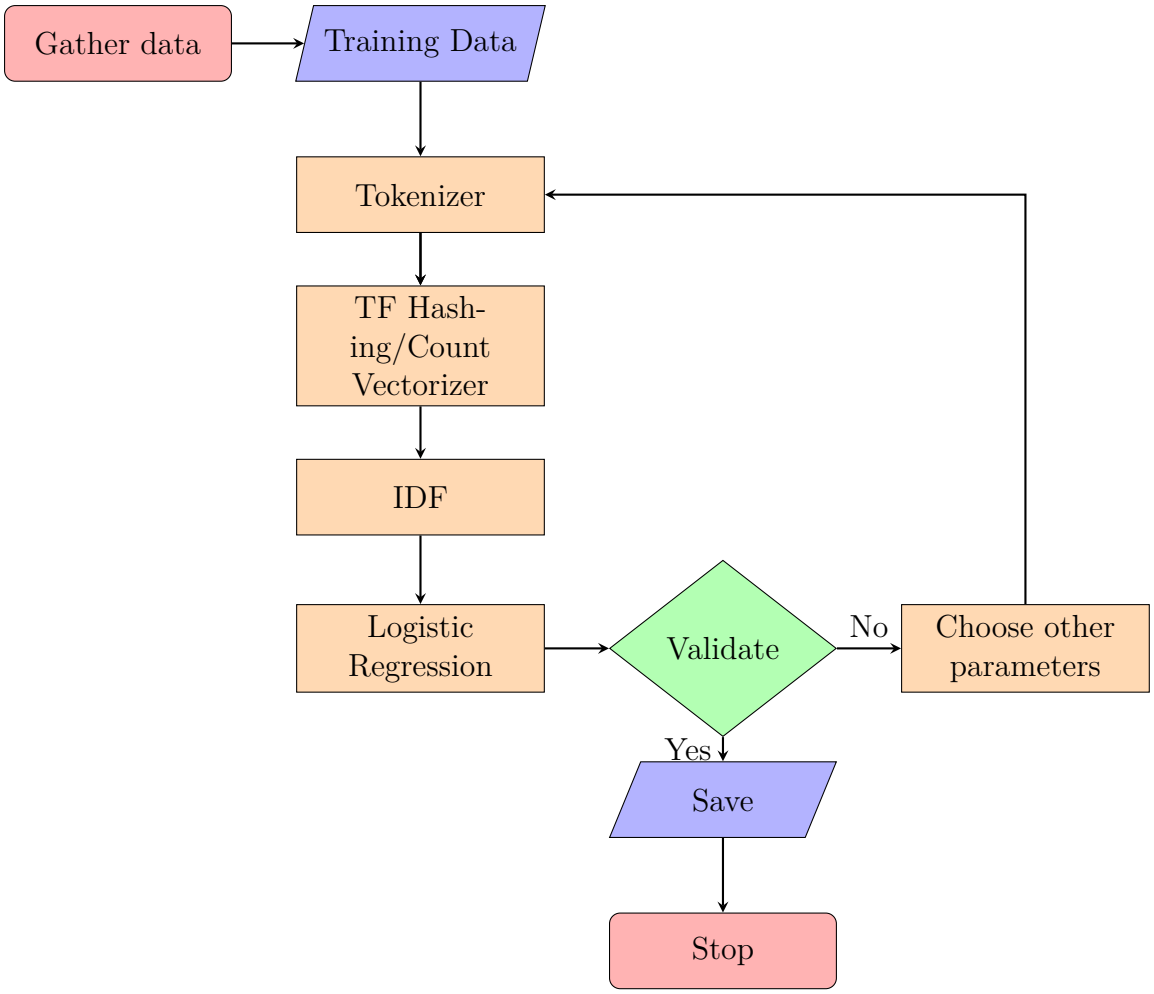
1. Collection of training dataset using Spark,
2. creating and training the MLLib classification model,
3. connecting the model to the stream,
4. making real-time predictions

3.1 Dataset construction

The dataset was created using the example script for data collection using Spark. It comes from the proxy server, which takes it from Steam. 2000 datapoints were recorded, in a form of game id, review text, review id and label.

3.2 Predictive model

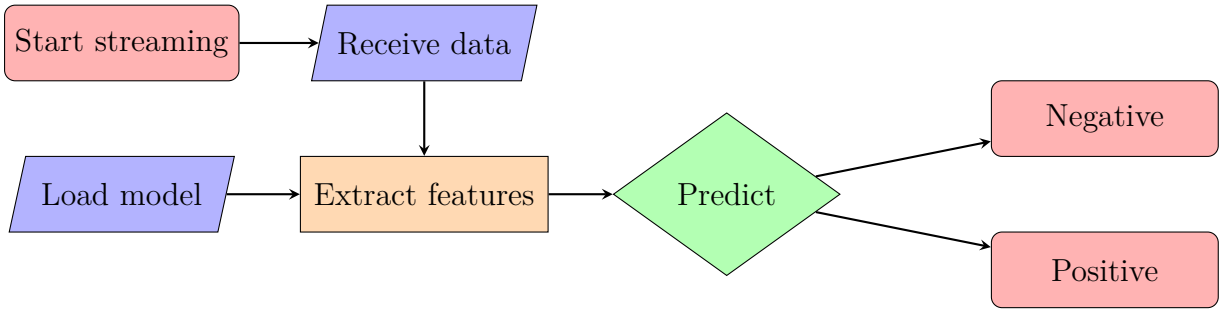
All functions related to the two models described below belong to pyspark.ml package. In order to ensure smooth workflow, we decided to use Pipeline function. The exact structure of the models is given below:



Our models have the accuracy scores of 0.79 and 0.78, so the performance is quite good. In the next part we are gonna use the one with TF Hashing, though it is a matter of changing one character to use the other model.

3.3 Setting up the stream

The following flowchart shows the basics:



3.4 Launching the stream

This part of the assignment is based on a local proxy file and stream-predicting example on Toledo. Our model has been trained in the last step. Here we only need to load the model. We have a data resource from RDD and we transform RDD data into dataframe which can be used in the model. Then we load our trained model in this process. Because

a single RDD can be divided into multiple logical partitions so that these partitions can be stored and processed on different machines of a cluster, we have to run our model in every partition of RDD. Some example output are as below, with formatting fixed to fit the rest of the document.

app_id	label	review_id	review_text	prediction
2339690	1	138710638	Good little game,...	1.0
2339690	1	138709412	Great game I like...	1.0
669330	0	138711423	Completely smoked...	0.0
669330	1	138710868	An easy to learn,...	1.0
2369390	0	138711840	Unfortunately, sa...	0.0
2369390	0	138711598	Finished the game...	1.0
2369390	1	138711402	yeee	1.0
2369390	0	138707666	I've been a FC fa...	0.0

Table 3: Example tablified results of streaming prediction

As we can see, the model is up and running, if the text is not complicated, then the model can predict the label. The delay observed is at $\tilde{2}$ seconds. Connecting and disconnecting to the stream works fine, can be done without any changes to the code itself.

3.5 Conclusion and Limitations

This model is above baseline, but not by much ($74\% vs 79\%$). More advanced techniques would be required to achieve better accuracy. Other thing is that we are running this in a controlled environment, where there are only three or four reviews per timestamp; all of them written in more or less understandable English. Below we can find an example of a reviews of the game Ultrakill, that our model would have "a bit of a trouble" to classify:

"Game is too good makes everything in my steam library look like **** in comparison, why can't more games be like this, its depressing."
"Doom is where boys become men Ultrakill is where men become femboys"
"contains scenes"
"don't go through the p door it will ruin your life"
"Pros: Best game ive ever played Cons: Cant keep the hampter no gay sex"

Try to guess the labels (answers will be in the reference section).

4 Assignment 4: exploring graphs with Cypher and Gephi

The aim of this assignment was to explore graphs using Cypher and Gephi. We worked with a data extracted from Twitch from December 2022 - April 2023. In the followings we present our aims, methods and findings and limitations.

4.1 General Context and Objective

On June 26, 1997, the first book of the Harry Potter saga from the pen of J.K. Rowling was published. Since then, this universe has enjoyed the success we know today, with hundreds of millions of fans worldwide, making J.K. Rowling one of the most famous authors of all time. But in recent years, her reputation seems to have been tarnished by a transphobia scandal[5].

It all started on March 19, 2018, when J.K. Rowling liked a transphobic tweet. She defended herself by claiming it was an accident - she had liked it while trying to take a screenshot. But other events continued to fuel the controversy. On December 19, 2019, J.K. Rowling endorsed Maya Forstater, an openly transphobic woman who had allegedly been fired for her views. More recently, on March 5, 2022, J.K. Rowling tweeted her opposition to a new law bill in Scotland to strengthen the rights of trans people. Since then, the Internet (and Twitter in particular) has gone on a rampage, accusing the author of transphobia.

Thus, in February 2023, when the game *Hogwarts Legacy* (a game based on the universe of the Harry Potter saga) was released, numerous LGBTQ+ activists called for a boycott of the game.

The goal of our research for this assignment[6], therefore, is to analyze data extracted from the video game streaming platform Twitch to determine whether or not the boycott actually occurred by examining the overall success of the game compared to other games, the game's connection to the LGBTQ+ community, and the situation of the LGBTQ+ community.

4.2 Data and methods

We obtain our data through memgraph query. We find all the streamers who tagged LGBT. Also, we focus on two games, Hogwarts Legacy and Elden Ring. We find all the streamers who play these two games respectively. Then we want to explore whether these games are LGBT-friendly. We also find, for each game, the streamers who play this game and also tagged "LGBT". In total, there are 66 tags[7] related to LGBT and 999 streamers tag these tags. There are 544 streamers playing Hogwarts Legacy and 217 streamers playing Elden Ring. Figure 9 shows that 7 streamers who play Hogwarts Legacy tag LGBT and 9 streamers who play Elden Ring tag LGBT.

[5] All the pieces of information were taken from the Internet

[6] Note that the aim is not to take a stand in the debate about J.K Rowling's possible transphobia.

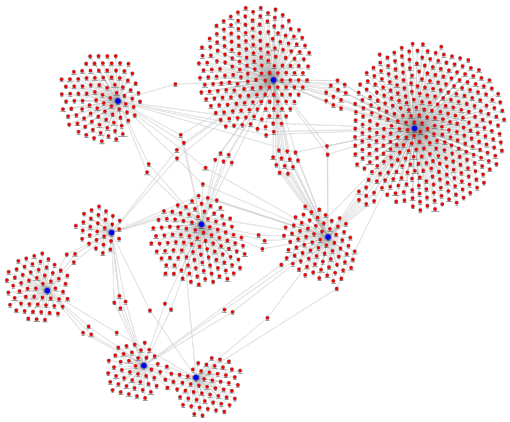
[7] Actually there are 100 tags, see section 4.2.1. When we wrote section 4.2.1, we added `toLower()` and discovered more tags.

4.2.1 Bird's eye view

Surprisingly, the LGBT community seems to be very fractured. There are 100 different tags used by streamers in the dataset where many only differ by virtue of capitalization. In table 7a, we listed the nine most popular lgbt tags. The remaining tags has too few streamers using it to be analyzed. In figure 7b, we can see that most streamers only uses one of the LGBT tags, and in figure 8 we show the recommendation map of streamers that both use one of the LGBT tags. Figure 8 strongly supports the hypothesis that the LGBT community is very fractured.

Tag	Streamers Tagging
LGBTQIAPlus	373
LGBTQIA	218
LGBTQIAplus	119
LGBT	106
LGBTQ	104
lgbtq	65
lgbtqia	62
lgbt	60
lgbtqiaplus	43

(a) Top LGBT related tags.



(b) Blue nodes are LGBT tags, red nodes are streamers.

Figure 7: A very fractured LGBT community.

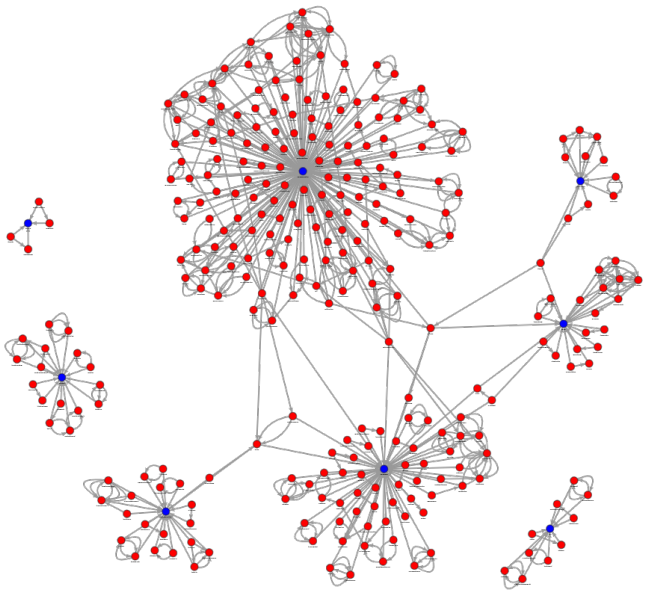


Figure 8: Recommendation map of streamers that both use the LGBT tag. Blue nodes are tags and red nodes are streamers. Edges between streamers are recommendations.

However, were one to step back and perhaps assume that these are not a single LGBT

Streamer	Average Views
breebunn	12515
blueandqueenie	11741
datadave	10433
pixelcircus	9367
pixelxkitten	9311

Table 4: Top streamers using LGBT tags.

community, in the sense that using the LGBT tags does not imply that a streamer necessarily is part of the LGBT community, rather supports/endorses it, then it makes sense that there are multiple isolated communities. In table 4, we showed the five most popular streamers using the LGBT tags. This hypothesis was formed after exploring the top streamer's twitch page.

A more nefarious line of reasoning would be to hypothesize that these streamers are using the LGBT tag for commercial reasons: to attract more views. However, we do not have the means to check this hypothesis, so we will drop it.

4.2.2 Deeper Analysis

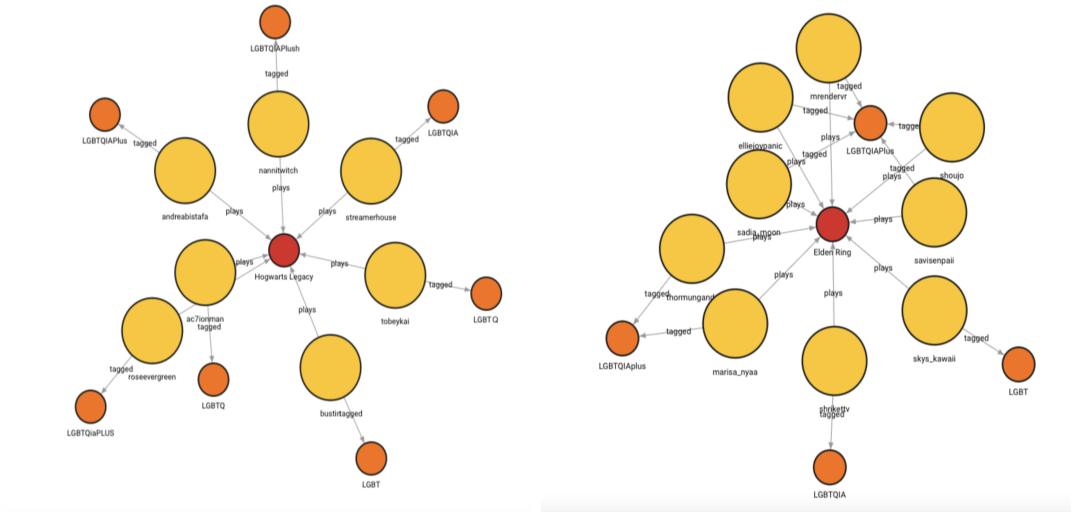


Figure 9: Hogwarts Legacy (left) streamers and Elden Ring (right) streamers who tagged "LGBT"

As it can be seen in Figure 10 there are not many more streamers who tagged "LGBTQ" and play with Elden Ring than Hogwarts Legacy. However, when we examine the popularity of the two games, it is clear that Hogwarts Legacy is more popular (Figure 9), hence in proportion to the players, there are fewer people who connect to LGBTQ and play with Hogwarts Legacy than Elden Ring.

In addition, those streamers, who tagged LGBTQ in any form and who play with Elden Ring, do not play with Hogwarts Legacy. Most streamers of Elden Ring tagged LGBTQIAPlus as opposed to Hogwarts Legacy, where the LGBTQ tag is more popular.

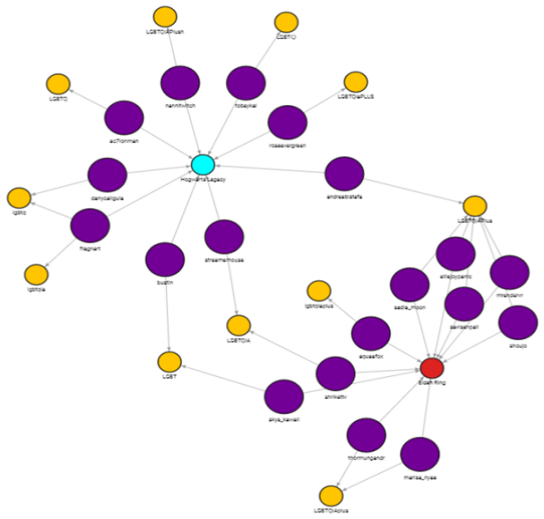


Figure 10: Hogwarts Legacy (blue) and Elden Rings (red) streamers who tagged "LGBT"

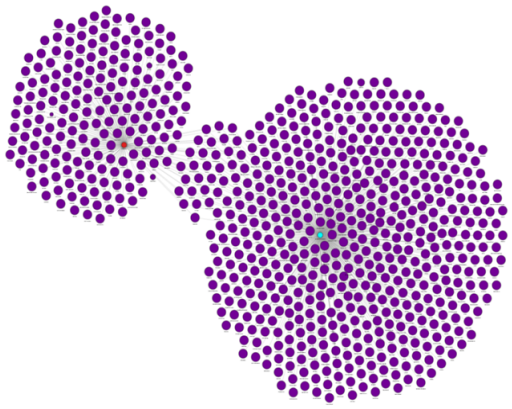


Figure 11: Popularity of Hogwarts Legacy (blue) and Elden Ring (red)

When taking a closer look at the description of the streamers, some interesting details are uncovered. A streamer amongst the Hogwarts Legacy players states: "Content outweighs morals :)". This may be a statement referring to the problem explained above. Also, a streamer who tagged Hogwarts Legacy is in fact StreamerHouse, which is an organization rather than a player.

There is no streamer in the trans community who plays with the Hogwarts Legacy game or even the Elden Ring game. This community seems to play with other games such as Hunt: Showdown or Fortnite; however, other non-games are also popular, e.g. chatting, ASMR or music as can be seen in Figure 12.

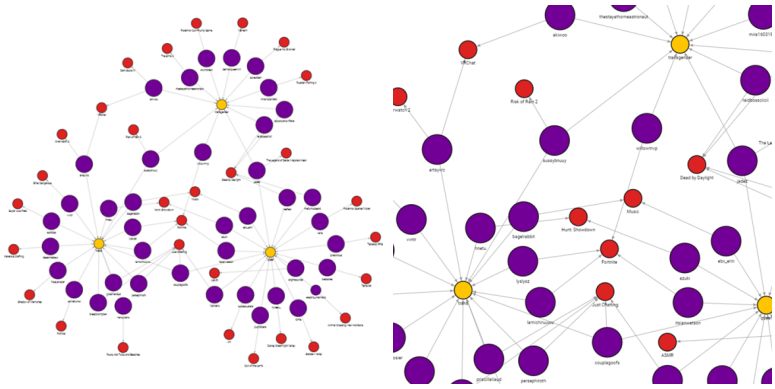


Figure 12: The trans community

4.2.3 Limitations

There are a lot of inherent limitations in the data. For example, we have no guarantee that people who tag themselves as LGBT, do not do it for profit purely. The data lacks many important factors, like how long was the average stream duration of a streamer, average viewership, interactions etc.

4.3 Conclusion

To summarize, our analysis suggests, that the boycott was a success. However, in today's economy, public opinion or amount of streams of the game doesn't matter - money does. While Twitch LGBT-tagged community did limit its exposure to that game, it might stem from the fact, that they usually look for a different type of games or content on that platform. In fact, once an author has put his or her work out there, it becomes everyone's. Also, J.K. Rowling doesn't have any involvement in this game but the game developers make use of the theme of Harry Potter. Thus some LGBT people also play this game. The sales results for this game are above average, topping the charts in many countries. There is no ground truth in those kinds of situations, so we tried to do it as objectively as it is possible.

5 References

Project's repository

Code for assignment 2 (Keras) on Google Colab

Code for assignment 2 (PyTorch) on Google Colab

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).

Labels for conclusions section for assignment 3: 0,1,1,1,0