KU LEUVEN

FACULTY OF ECONOMICS AND BUSINESS

Report

Advanced Analytics in a Big Data World

GROUP 2

Arthur De Rieck - r0710355

Avelyn Araya - r0865644

Diviji Pherwani - r0909423

Joana Anselmo - r0910743

Jolien Covens - r0744356

Mitja Mandić - r0875362

Professor Seppe vanden Broucke

Academic year 2021-2022

29th of May 2022

Overview

Assignment 1: Predictive Model on Tabular Data

1. Introduction

The first assignment consists of building a predictive model to predict a churn target (customer retention setting). The program used to preprocess the dataset and construct the model was Python. All the images presented in this assignment have Python as the source.

There were given six datasets to use (three train sets and three test sets). The dataset mainly used for the development of the model is **train_month_3_with_target.csv**, even though the others will be also used for comparing results and discovering hidden inferences. The dataset used for testing the model is **test_month_3.csv**.

The first step was to import the necessary packages to execute the analysis and visualizations of the data. In this case, the packages were: **pandas**, **numpy**, **datetime** (to get the current year value), **matplot.pyplot**, **seaborn** and **graphviz** (the last three for visualization).

Then, a function was defined to convert a data array back to a dataframe and restore the datatypes. It only restored the datatypes bool, int64, and string. **Input:** *data_array:* an array that is a subset of the original dataframe og_df. All features of og_df should be included in the data_array, except for the ones in exclude; *og_df:* the original dataframe; *exclude:* an array containing the names of the features that are present in og_df, but not in data_array. **Returns:** the given array data_array as a dataframe.

2. Preprocessing

2.1 Load the data

The data from the dataset **train_month_3_with_target.csv** was loaded and printed the dimensions of the first 5 instances. In total, it has 63697 rows and 40 columns.

2.2 Missing Values

In this step, it was verified which features had missing values, being the following: *customer_since_all* (234), *customer_since_bank* (249), *customer_occupation_code* (2002), *customer_education* (47125), *customer_children* (23364) and *customer_relationship* (14899). Of those values, 61784 were negative and 1913 were positive.

It made sense to drop *customer_occupation_code*, *customer_education*, *customer_children*, *customer_relationship*, given that these do not have an immediate link to churn probability. Those were also the features with the most missing values (missing for 3%, 74%, 37% and 23% of the observations, respectively), indicating that it might be difficult to obtain this

information. Since *customer_since_all* and *customer_since_bank* are dates, reasonable imputation would be difficult, so these features were also dropped.

2.3 Data Types

The data type of each feature was checked and *client_id*, *customer_birth_date*, and *customer_postal_code* were converted to string. Also *visits_distinct_so* and *visits_distinct_so_areas* were converted to integers. Additionally, the binary coded features were converted to type boolean instead of integer or float (*customer_gender* has values 1 and 2 which should be converted to 0 and 1 before changing the datatype to boolean).

2.4 Transforming Features

In this step, the feature *customer_birth_date* was converted to a new feature *customer_age*. This was done based on the birthyear. Then, when doing a summary of the numerical features, it was observed that the maximum value for *customer_age* was 158 years. To solve the issue, a condition to filter out instances was established where ***customer_age* > 100** (because it is rare that customers will have an age superior to 100 years old), which led to a total of 242 observations filtered out (0,38%). Finally, the feature *customer_birth_date* was dropped out of the dataframe.

3. Looking at Old Data

First, the **train_month_2.csv** and **train_month_1.csv** datasets were preprocessed the same way as the initial dataset.

Then, an analysis was done to search for people who were possible churners among those who turned off a feature. The conclusions were that there were people who turned off *homebanking_active*, few people switched off insurance, and nobody who canceled insurance also canceled homebanking (630 turned off ('homebanking_active', 'has_current_account'), of those 23 churned and 167 turned off homebanking_active, of those 6 churned).

Another analysis conducted was to sum across rows to find which people are closing their products and find possible churners. 45 people whose number of products decreased churned and 1059 did not.

A graph was elaborated to demonstrate how the number of visits is distributed among churnerns and nonchurners, concluding that customers that are churners (bar 1) visit the bank more than non-churners (bar 2).

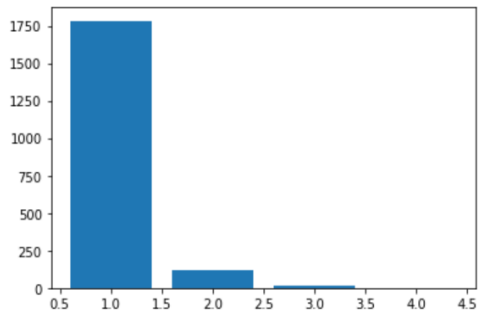

Figure 1 – Number of visits by churners and nonchurners

4. Split the Data

Returning to the **train_month_3_with_target.csv** dataset, the features and target were separated and converted to arrays (necessary for using sklearn). Then, the data was split into training (75%) and validation (25%) set.

5. Feature Selection

5.1 Correlation

In this phase, the train dataset was converted back to a dataframe and the training target was added to the training data. The *client_id* feature was also dropped.

Then, only numerical features were selected and converted from boolean back to numerical so they could be included in the correlation matrix. Next, the correlations with the *target* (Appendix 1) were analyzed and for the features with a *has_[...]* and *bal_[...]* that are related (and other similar relations), only the *bal_[...]* feature was kept. Out of *visits_distinct_so* and *visits_distinct_so_areas*, only the first was kept because its correlation with the target is higher.

Since the predictive ability of the model was rather low, a higher correlation threshold was used. Based on that, the correlation of the variables *homebanking_active*, *bal_mortgage_loan*, *cap_life_insurance_decreasing_cap*, *visits_distinct_so*, *prem_fire_car_other_insurance* and *customer_age* was calculated to see which were the three features with the strongest correlations with each feature.

With that analysis, it was decided that *prem_fire_car_other_insurance* could be dropped since it is correlated to *visits_distinct_so*, which is correlated to more features. *Customer_age* and *homebanking_active* could also be dropped for *cap_life_insurance_decreasing_cap*. V*isits_distinct_so* and *cap_life_insurance_decreasing_cap* were also correlated. In summary, only *cap_life_insurance_decreasing_cap* was kept from the list **corr_features** and *visits_distinct_so*.

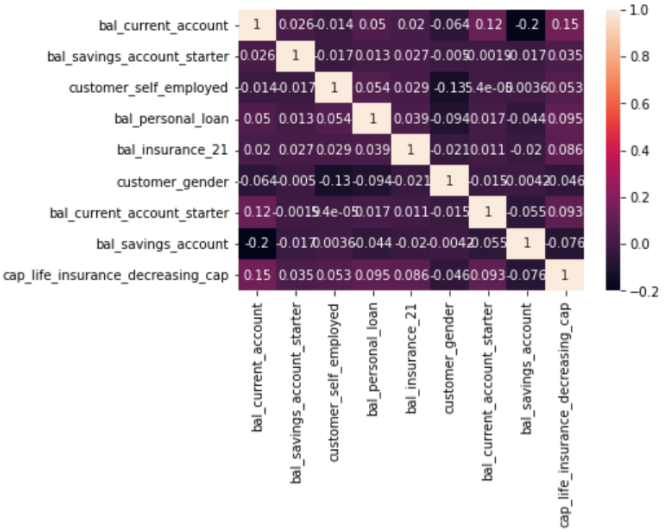

Figure 2 – Correlation Heatmap

5.2 PCA

First, the train dataset was converted back to a dataframe (44418 rows × 32 columns) and restored the data types of the features. Then, only the features kept on the correlation section that are numerical were selected. It was converted back to an array (for sklearn).

Next, the data was scaled so all the features would be equally important by PCA and the PCA was applied with the number of components that explains at least 80% of variance, which resulted in 8 components.

After that, the PCA was calculated with the maximum number of components (a total of 11), which 4 of them explained 51% of the variance.

```
percentage of variance explained by each component:
[0.21 0.11 0.1  0.09 0.09 0.09 0.08 0.08 0.06 0.06 0.03]

total percentage of variance explained by the first n = 4 components:
51.0 %
```

Figure 3 – PCA with all the components

Based on that, a scatter plot was constructed with the principal components which, as it can be seen in Figure 2, are components 1 and 2.

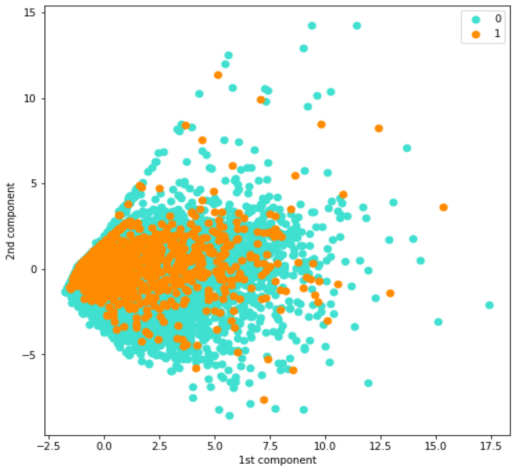

Figure 4 – Scatter Plot of Components 1 and 2

Then, the loadings of the first 4 components were computed, and it can be observed that only the first component has 2 variables with loadings superior to 0.7 (*bal_mortage_loan* and *cap_life_insurance_decreasing_cap*). This analysis does not offer solid conclusions of how to group the variables into components, since the loadings have such small numbers (some of them are inferior to 0.5).

loadings on first 4 components:

	1	2	3	4
bal_mortgage_loan	0.776575	0.128718	-0.303914	0.235096
cap_life_insurance_decreasing_cap	0.779471	0.140642	-0.352144	0.180705
bal_savings_account	-0.243248	-0.472840	-0.473230	0.362440
bal_current_account	0.354356	0.290009	0.591491	0.190100
bal_personal_loan	0.300844	-0.205205	0.148720	0.012954
customer_age	-0.418683	-0.356001	0.304002	0.460604
visits_distinct_so	0.584404	-0.453832	0.140039	-0.036980
prem_fire_car_other_insurance	0.381345	-0.580205	0.269996	0.141519
bal_current_account_starter	0.202558	0.311069	0.281948	0.254013
bal_savings_account_starter	0.139784	-0.193625	0.166453	-0.267383
bal_insurance_21	0.251995	-0.254571	0.012490	-0.635356

Figure 5 – Loadings of PCA 4 components

6. Modeling

6.1 Classification Tree

In this step, only the features given by *feature_set* (defined in the section 'Correlation') were selected from the training and validation set. Then the data was converted back to arrays so it could be used with the sklearn methods, and the targets were converted from boolean to integer.

After that, the goal was to predict the targets of the validation set, by putting the probabilities, predicted target, and true target in one dataframe and using the probabilities to rank the instances (keeping only the top 250). Then, the confusion matrix and AUC (=0.56382979) were calculated.

```
confusion matrix:
[[171  64]
 [  9   6]]

true positives:  6
false positives: 64
true negatives:  171
false negatives: 9
```

Figure 6 – Confusion Matrix of Classification tree

6.2 Logistic Regression

With logistic regression, the previous method was applied. To avoid Misclassification costs, a grid search for optimal weights was performed for each of the two classes. At each value in parameter "weights", a cross validation procedure was applied and then selected the parameters with the highest score according to the "roc_auc" setting. 5 folds for each of 200 candidates (with a total of 1000 fits) were fitted. Best score: 0.7133453386657517 with param: {'class_weight': {0: 0.024874371859296484, 1: 0.9751256281407035}}.

```
confusion matrix:
[[15214   233]
 [  461    17]]

true positives:  17
false positives: 233
true negatives:  15214
false negatives: 461
```

```
confusion matrix:
[[15225   222]
 [  450    28]]

true positives:  28
false positives: 222
true negatives:  15225
false negatives: 450
```

Figure 7 and 8 – Confusion Matrix of Logistic Regression (before and after class weights)

6.3 Random Forest

For Random Forest, there were three options to preprocess the data. The first option was the one used in section 2.2 Missing Values.

For the second option, indicators were used for the features with missing values. Additionally, the *customer_since_all* and *customer_since_bank* features were transformed to store the number of years instead of the start date. The missing values for features in the train and test sets were then replaced by the most frequent values in the train set.

For the third option, the indicator variables were again used, as well as the transformation for the *customer_since_all* and *customer_since_bank* features. For *customer_since_all*, *customer_since_bank* and *customer_children* features, the values used for imputation were determined based on age. This means that a table was made for each distinct age, being the most frequent value of the feature for instances with that age. When an instance in the test set has a missing value for one of these features, the value from the table is used that corresponds to the age of that instance, or, if there is no row in the reference table for this age, the value for the age closest to that of the instance is used. For the other features with missing values, the most frequent value in the train set was used. This was the option used to preprocess the data for the model.

Then, the *customer_gender* feature was recoded to 0 and 1 instead of 1 and 2. For the *customer_postal_code* feature, only the first value was kept, resulting in a categorical variable with 10 levels. The categorical variables were transformed into binary variables with a one-hot-encoder.

After that, possible scoring functions were defined. The scorer that was eventually used calculates the number of churners present in the top 250 predictions. The features and target were separated and converted to arrays (necessary for using sklearn). The data was split into training and test set (75-25) before the preprocessing, with the imputation values of the train set being used for the test set as well. *client_id* was dropped because it should not be used for making predictions.

Next the parameters for the model were determined through cross-validation. The options for the final training of the model are given below. This was the model submitted on the leaderboard. The confusion matrix of the model was also calculated.

- 10 fold
- preprocess('train_month_3_with_target.csv', 'test_month_3.csv', 3)
- own_scorer_02 (after small change)
- 'class_weight': ['balanced', {0:1, 1:500}, None],
- 'max_features': ["sqrt", 0.05, 0.1, 0.2, 0.3, 0.5],
- 'max_samples': [50, 100],
- 'n_estimators': [500, 750, 1000]
- best: `{'class_weight': 'balanced', 'max_features': 0.2, 'max_samples': 100, 'n_estimators': 500}`
- number of churners in top 250: 37
- total number of churners: 478
- percentage of top 250 that are churners: 14.8 %
- percentage of churners that is in top 250: 7.7 %
- `out09`

Figure 9 – Best Random Forest, Output 09

```
confusion matrix:
[[15234   213]
 [  441    37]]

true positives:  37
false positives: 213
true negatives:  15234
false negatives: 441
```

Figure 10 – Confusion Matrix of Random Forest

In order to get some insight into how the model makes decisions, partial dependence plots of the features were analyzed. The features that stand out in the partial dependence plots have the strongest influence on the churning probability by themselves. Some features that have a constant partial dependence might have a stronger effect if interaction with other features is taken into account.

According to the model, the probability to be a churner seems to increase with age, being largest between roughly 40 and 60, after which it decreases to a level below even the starting level. Features related to insurance seem to have a similar effect to each other, namely the higher the value, the higher the probability. The exceptions are *cap_life_insurance_fixed_cap* and *bal_insurance_23*, which stay more or less constant, possibly due to an interaction effect. Feature *cap_life_insurance_decreasing_cap* seems to have the greatest effect, eventually capping off at a value of 0.0451.

Higher values for mortgage and personal loans also increase the churn probability. For mortgage loans, a maximum probability is reached around approximately 200,000, while for personal loans the maximum is reached around 35,000. The balance of the savings account has a strong effect on the churn probability, namely a low balance is connected to a high probability. The higher the balance, the lower the probability drops. The descent slows down up to a balance of 20,000, after which the probability becomes more or less constant. The balance of the current account has the opposite effect to that for the savings account: the higher the balance, the greater the probability. The evolution for the balance of the current account is however less steep and also exhibits a peak just before a value of 5000, after which the probability drops before rising again at a slower pace. The balance of a starter current account shows a similar behavior as that of a current account, although without the aforementioned peak.

The features related to whether the customer has children are more or less constant, except for the indicators for customers that have children who are grown-up or children who are in preschool. For these instances, the churn probability is higher. Those with insurance seem to have a lower churn probability, although having a life insurance with decreasing capital seems to result in a higher probability. Instances with loans, either personal or mortgage, seem to have a higher probability as well.

Education codes 1, 2, 5, and 6 seem to increase the churn probability, whereas code 3 seems to decrease it and code 4 has a rather constant plot. The indicator for a missing value for the education code seems to lower the probability. The features related to postal code are more or less constant, except for those starting with a 6, for which the churn probability seems to be higher. When it comes to occupation, those with code 9 have a lower probability, while those with code 4 and those who are self-employed have a higher churn probability.

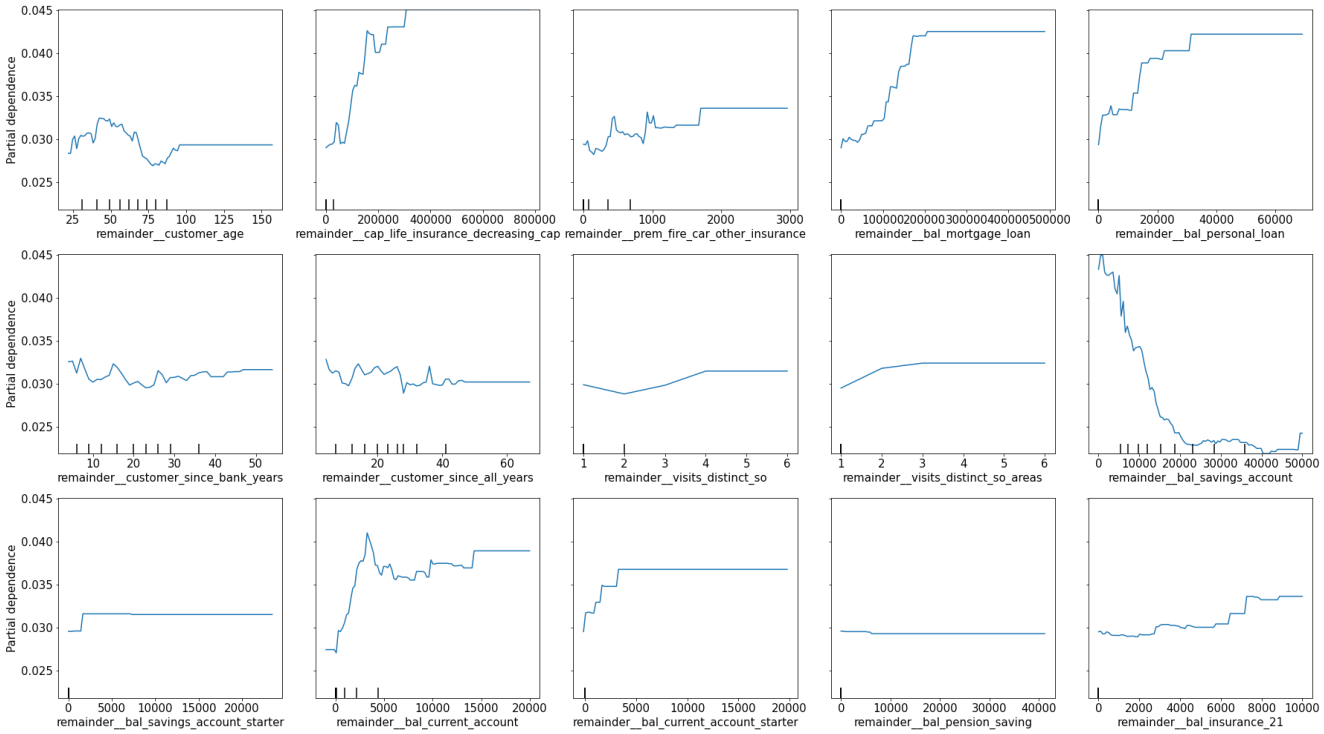

Figure 11 – Partial dependence plots for the numerical features with the strongest effects.

The scoring of the models makes sense from a practical perspective, since the results of the model need to be actionable. You do not necessarily care about the score on the entire dataset, since you will only be able to address a certain number of people, here taken to be 250. However, the scoring does not take into account how important a client is. Finding churners who have a lot of money invested in your bank through accounts, insurance products, loans, etc., is probably more important to you than finding those who, for example, only have a current account. It would therefore be interesting to add a penalty to the scoring if a high-value client who is a churner was not included in the top 250.

7. Conclusion

Based on the results obtained, the best model is the one generated using Random Forest. Of the top 250 most likely churners 37 were true positives. In terms of confusion matrix, it is the model that presents the highest number of true negatives and true positives. The public score of the model was 41, with the final score being 25. The model also had a decent AUC value of 0.71. This means that a randomly chosen churner will have a 71% chance to be given a higher score by the model (churn probability) than a randomly chosen non-churner.

When using cross-validation to determine the best parameters for the random forest model, the selected set seems to be a bit unstable. This means that small changes in the model can lead to a different set of parameters being chosen and since a random forest model is stochastic, small

differences are always present due to randomness. The analysis of the individual partial dependence plots suggests that the probability of churning is generally rather low and that the change in probability due to different values of the features is small. This is not surprising considering that only a very small subset of the data are categorized as churners. However, since the overall churn probability and influence of the feature values are rather small, a slight change in the model or data could result in a different ordering of the instances based on churn probability.

Since the choice for the best parameters can be influenced by small changes in the model or data, overfitting on the training data is possible even with the use of cross-validation and a test data set, since a slightly different training set could result in different parameters being selected. While most of the learned classification rules that present themselves in the partial dependence plots do seem to make sense, the exact feature values for which the probability increases or decreases and how strongly the probability is influenced might be more specific to the training data, which negatively impacts the ability to generalize to unseen data sets.

In conclusion, while the model seems capable of finding reasonable rules to determine the churning probability, the large discrepancy between the number of churners versus non-churners in the data results in an overall low churn probability and a rather small influence of the feature values on this probability. This makes the model more sensitive to overfitting on the training data, which given the drop in obtained score does seem to be the case. It would be interesting to look into sampling approaches to balance out this discrepancy and as a result bias the model towards detecting churners. This could result in greater differences between parameter sets when the model is evaluated during cross-validation, and lead to a model that generalizes better to unseen data.

Assignment 2: Deep Learning on Images

1. Introduction

The second assignment consisted of building a neural network capable of detecting a swimming pool(s) in a static image.

A dataset of 14912 images, in a zip file, and their annotations in a single JSON file were provided. The images were annotated as polygons using Overpy and the Overpass API together with the Mapbox static maps API. The **metadata.json** file consisted of the following data:

1. **name of the image.png**, which are also the geographic coordinates of the center.
2. **has_pool**, a boolean variable which indicates if there is a pool in the image or not.
3. **center**, a dictionary with the geographic center coordinates.
4. **bounds_lat_lon**, a list that contains the geographic coordinates of the bounding polygon.
5. **bounds _x _y**, a list that contains the cartesian coordinates of the bounding polygon.

This focus is on predicting a bounding box around the pool. For this two models were implemented: You Only Look Once v5 (YOLO) and RetinaNet (Retina) and compared in terms of performance.

2. Preprocessing

2.1. Importing packages

The following python packages were used: **Pytorch, torchvision, numpy, PyYAML, pathlib, numpy, json, OpenCV, pandas, os** and **sklearn**

2.2. Characteristics of the data

For every image in our dataset, which were all the same size, exactly one bounding polygon is present in the middle of the image. This means there were no images with zero or 2+ pools labeled, even though in reality some images did have more than one pool. The labeling was not perfect as sometimes things that do not look like a swimming pool (see as example Figure 12) were labeled. Other images had their pools covered by either a roof, foliage or were shaded (see as example Figure 13) or their polygons were shifted (see as example Figure 14). However, the assumption was made that the majority of the images were labeled correctly and therefore no images were dropped from the dataset.

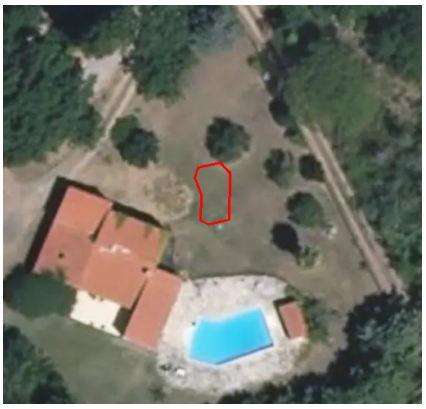

Figure 12 – Image and its respecting label polygon

Figure 13 – Image and its respecting label polygon

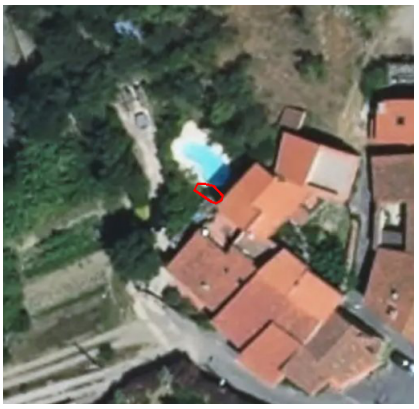

Figure 14 – Image and its respecting label polygon

2.3. Transforming the metadata

All metadata was stored in a single json file with multiple levels of dictionaries and lists. To be able to more easily access the data for a specific image, **metadata.json** was read into a pandas dataframe. This dataframe was transposed so every image had its own row.

2.4. Polygon to bounding box

Since both YOLO and Retina need an upright bounding box as their input, the polygons had to be transformed into a box. To make sure the entire pool was captured in the new bounding box, four new lists were created: *datax, datay, datamx, datamy*. These lists contain the maximum and minimum of both the x and y coordinates by looping over all x or y coordinates in the *bounds_x_y* dictionary, saving the value if it was more extreme than the current extremum and appending the extremum for each image to the list. To avoid boxes from exceeding the limit of the image, the minima and maxima were capped at 0 and 512 using max and min functions respectively. Otherwise an "out of bound label" error would pop up during training.

Figure 15 – bounding box + polygon

2.5. Darknet

To be able to run YOLO, the input needs to be converted into the Darknet format. This means that every image has to have a seperate .txt file with the following information: *category_idx, x_center, y_center, bbox_width and bbox_height,* with the last 4 normalized to be between 0 and 1. Multiple rows with each containing a single bounding box are allowed but not relevant in our case. These variables were calculated using the previously constructed *datax, datay, datamx, datamy* lists and appended to the dataframe. At this point separate training and test sets were defined using a 70/30 split.

A custom function was then written to copy the images to specific directories and write the separate .txt files for each image.

2.6. PASCAL VOC

To use RetinaNet, the data needed to be labeled in the PASCAL VOC format. This means each image has a separate .xml file which contains the path to the image, the folder it is contained in, name of the image, source of the database (by default is unknown), the width, height and depth of the images, if it is segmented (by default is zero), the name of the class (*swimmingpool*), if it is truncated or if it is difficult (to both by default is zero) and the bounding box defined by the minimum x and y and the maximum x and y position of the bounding box.

A directory named *Pool* was created, containing three subfolders: *JPEGImages*, that contains the images from the .zip file as .jpg format, *Annotations*, which contains the labels for each image in PASCAL VOC format and *ImagesSets*, which contains the list of the all images reserved to training and validation (12675) and their respective independent lists (images for training: 10438, images for validation: 2237). The decision of the amount of images for training, validation and testing was made following a 70/15/15 composition.

3. YOLOv5

3.1. Introduction to YOLO

You Only Look Once is a one-stage convolutional neural network which firstly divides the image into a grid with cells of equal dimension. It then draws B bounding box(es) within each grid cell. A box can exceed the limits of the cell but the center has to be in the grid cell. Next, it eliminates redundant bounding boxes by taking the bounding box with the highest predicted probability if two or more bounding boxes have an IoU above a certain threshold and the same predicted class. As the name suggests this model only passes once over the data and therefore performs very well in terms of prediction speed. (Bandyopadhyay, 2021)

3.2. YOLO architecture

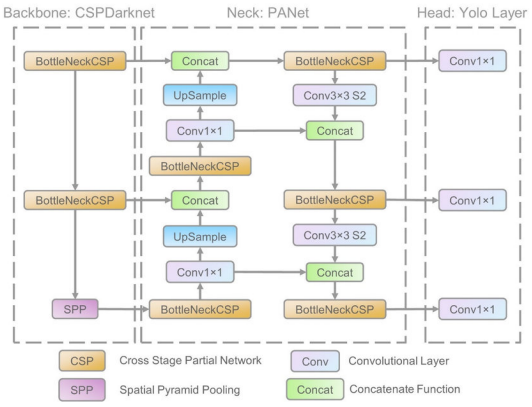

Figure 16 -- Yolo Architecture (Xu, Lin, Lu, Cao, Liu,2021)

As figure 16 shows YOLO's architecture has 3 parts: backbone, neck and head. Here, the backbone consists mainly of Cross Stage Partial networks who are responsible for retrieving the information out of the images. Next, the neck aims to construct feature pyramids using PANet. Lastly the head performs the actual predictions. YOLO has a different convolutional layer for small, medium and big objects. (Patel, 2020)

3.3. Training YOLO

The first step was to set up the environment correctly. To get access to a GPU, Google Colab was used with Pytorch and YAML. To access the correct models, https://github.com/ultralytics/yolov5 was cloned into the notebook. Since we only want to detect one class and not 80 like in the default pretrained model, two custom .yaml files were downloaded to configure the dataset and model correctly. The model is a pretrained model called YOLOv5L.

Using train.py, the model was trained in batches of 4 images and 20 epochs. The results were cached and weights were randomly initiated. The image resolution was reduced from 512x512 to 256x256. This allows for faster training since this enables google colab to cache the data. Training took 2 hours.

3.4. Evaluating YOLO

An evaluation tool is included in the github repo that was cloned. This returns the following graphs:

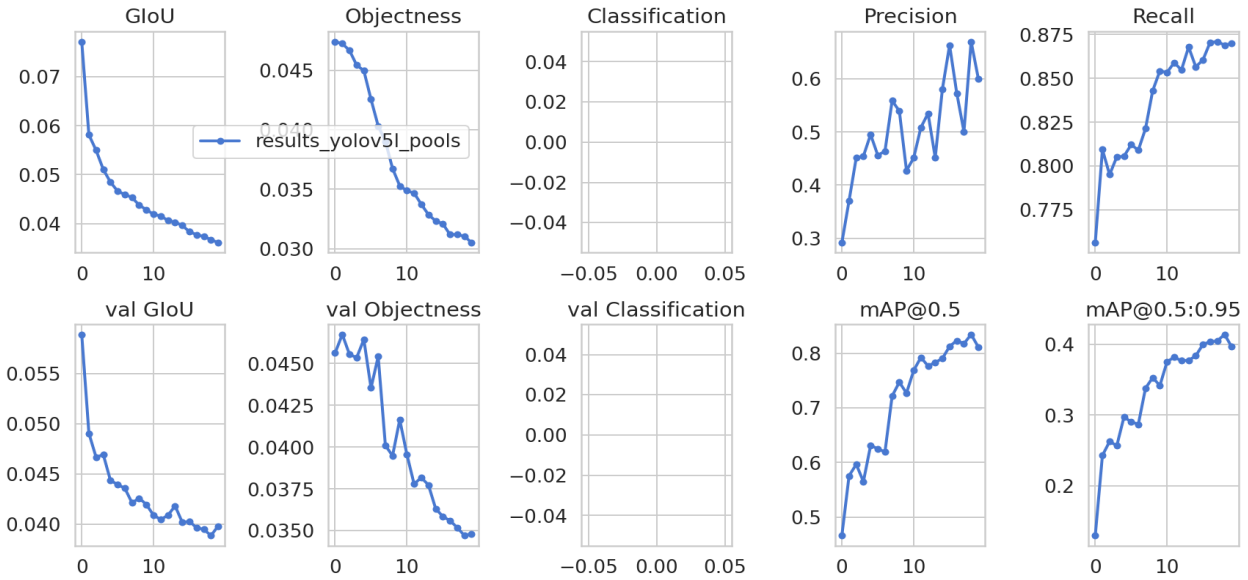

Figure 17 – results of yolov5l training

Since there only was 1 class present in the dataset the classification loss is constant and zero for both the training and test set. Generalized Intersection over Union is IoU corrected by a factor that includes the smallest box covering both the true and predicted box and is therefore a lower bound of IoU (Rezatofighi et al, 2019). Objectness is Binary Cross-Entropy and represents the confidence the model has that there is an object within a box. Both GIoU and Objectness are included in the objective function YOLO is trying to minimize. The reported performance measures are Precision, Recall, mAP@0.5 and mAP@0.5:0.95. A box is considered positive if the model returns a confidence higher than a certain threshold when the model is asked to classify it. Precision is the amount of true positives divided by the amount of true and false positives. This

value fluctuates but peaks at 67%. Recall is the amount of true positives divided by true positives and false negatives. Recall is starting to stabilize at around 87%. mAP@0.5 (mean average precision with IoU @threshold of 0.5) peaks at 83% and mAP@0.5:0.95 (IoU threshold ranging from 0.5 to 0.95) peaks at 41%. In general they are all still improving (except for the last epoch) so performance could have been improved by adding extra epochs.

The models were asked to give some predictions on an example set of 100 images with confidence larger than 0.2. Despite having only seen images with exactly 1 pool, it is able to detect 0 or multiple pools in a single image, see figure 18 below. However, it seems to have over-fitted on pools being in the center. Figure 19 shows that when being given an example of a blue bus it gives a 25% confidence for a pool being on the side of the bus. In figure 20, YOLO predicts with 60% confidence that there is a pool in the middle but misses the much more obvious pools in the top left corner. This could have been avoided by enhancing the dataset with cropped images where the pools were not located in the middle. Figure 19 proves that YOLO can handle differently sized images.

Figure 18 – 42.6903244_3.0298364.png

Figure 19 - bus image

Figure 20 - 43.5786617_1.3284555.png

4. RetinaNet

4.1. Introduction to RetinaNet

RetinaNet is a one-stage object detection model that utilizes a *focal loss* to address class imbalance during training, due to an extensive background class. Its architecture consists of a backbone net (for this assignment ResNet50), which is responsible for calculating a piramidal feature map, and two specific subnetworks: one dedicated to perform the object classification and the other performs the bounding box regression. The *focal loss* is used to obtain the loss of the classification subnet (Lin et al., 2017.)

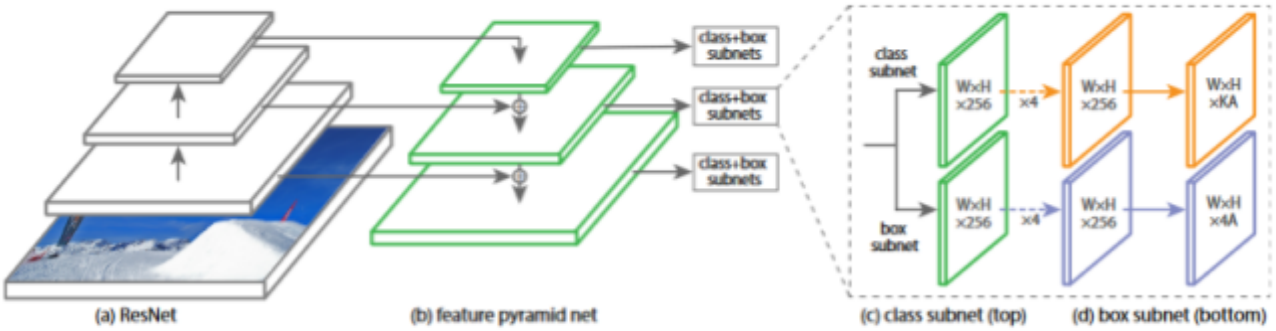

Figure 21 - An illustration of the RetinaNet architecture (Lin et al., 2017)

4.2 Training RetinaNet

The implementation of keras-retinanet from https://github.com/jaspereb/Retinanet-Tutorial was used (that used https://github.com/fizyr/keras-retinanet implementation as base) to prepare the data, create the model, train it and test it. First, the repository was downloaded to the same level as the *Pool* folder.

The pascal_voc.py script in the preprocessing folder of keras-retinanet was rewritten so it contain only one class *'swimmingpool'* and then run the train.py script in the bin folder (plus a few redirectioning paths) with 50 epochs and 100 steps per epoch. The ResNet50 architecture started with ImageNet pretrained weights. The model has a dynamic learning rate and an early stopping class from keras to deal with minimization and overfitting issues respectively.

Because of the lack of a functional CUDA GPU, the training was done with a CPU which made the training take considerably more time, it took 3 days.

4.3 Evaluating RetinaNet

Tensorboard was used to evaluate the progress of the training. Figure 21 shows, per epoch, the mean Average Precision at IoU threshold of 0.5 (mAP), the classification loss of the classification subnet, the regression loss of the regression subnet and the dynamic learning rate. Since our case only has one class, the mAP of the class is the same as the mAP of all classes. It can be noted that the mAP consistently improves across epochs while the losses decrease. Once the model reached a plateau, it lowered its learning rate. This happened around epoch 20 where mAP only rose a little, while the classification loss got worse. Due to insufficient memory issues, the model stopped training in epoch 30. Both losses do not indicate a convergence so maybe if the model was able to train more it could have achieved a better performance.

For this report, the model train obtained after the 27th epoch is considered as the best model since it is the model with smallest losses and biggest mAP.

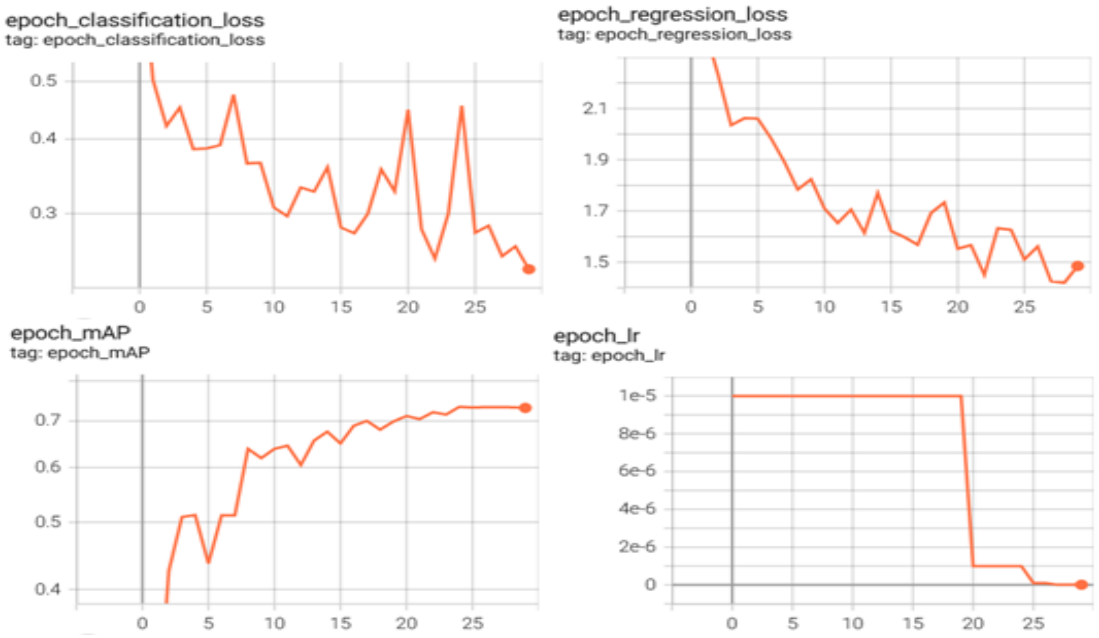

Figure 22 - results of RetinaNet training

This obtained model was tested on 100 images out of the testset (which contained 15% of the images), and a few original photos, to test the performance of the model in a new set. To obtain these results, testDectector.py script provided by *jaspered* repository ran the model on these photos using a threshold equal to 0.5 (0.2 for the original photos). Around 11 images didn't return a pool when it should have, most of these images had its pool covered by shadow or a roof. But we know Retina can detect some of these hidden pools, as can be seen in figure 23. Just like YOLO it had a tendency to identify objects in blue buses with around 20% of confidence (see figure 24). Also, the model does have difficulties finding pools on the edges, in Figure 25 for example, Retina could only identify the ones closest to the center which indicates that the model is considering the position of the pool in the image as a feature. Just like YOLO, the bus image is a good example of how Retina is able to handle differently sized images.

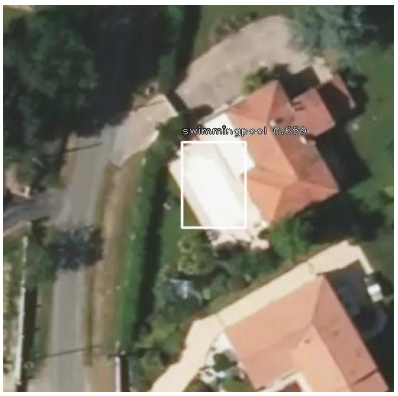

Figure 23 - 3.6956419_1.5083703.jpg

Figure 24 – Detection on a bus with threshold at 0.2

Figure 25 - 42.6848574_2.8048516.jpg

5. Conclusion

We can see that the YOLO model selected in this report was superior to the RetinaNet model selected here. 83% vs 73%. This conclusion is based on the training time and mAP@0.5 metric, but the former can be improved if used tools like Colab or reducing the size of the image.

We can see that both models overfitted the bounding box position, concentrating on the center of the images. To avoid this we can add training images with box labels that are not in the center. We could also enhance the dataset by adding a cropped version of the dataset where pools are not in the middle to the training set. Both models resize the model to their convenience before training, so a training set with images with different sizes should not be a problem. Finally we could improve the results of both models with more training time.

6. References

Bandyopadhyay, H. (2022, May 6) *Yolo object detection*. Retrieved March 21, 2022 from https://www.v7labs.com/blog/yolo-object-detection

Curiously. (2020, June 27). *Object detection on custom dataset with yolo v5 y using pytorch and python.* Retrieved May 14, 2022, from https://curiousily.com/posts/object-detection-on-custom-dataset-with-yolo-v5-using-pytorch-and-python/

Xu, R., Lin, H., Lu, K., Cao, L., Liu, Yi. (2021). *A Forest Fire Detection System Based on Ensemble Learning. Forests*. 12. 217. 10.3390/f12020217.

Rezatofighi, H.,Tsoi, N., Gwak, J. Y., Sadeghian, A., Reid, I., Savarese, S. (2019) *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression*

Patel, M. (2020, July 1). YOLO V5 --Explained and demystified, Retrieved May 24, 2022 from https://towardsai.net/p/computer-vision/yolo-v5%E2%80%8A-%E2%80%8Aexplained-and-demystified

Ultralytics. (s.d.) *YOLOv5 in PyTorch*. Retrieved May 14, 2022, from https://github.com/ultralytics/yolov5

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollar, P. (2017). Focal loss for dense object detection. *2017 IEEE International Conference on Computer Vision (ICCV)*. https://doi.org/10.1109/iccv.2017.324

Fizyr. *Keras RetinaNet.* Retrieved May 22, 2022, from https://github.com/fizyr/keras-retinanet

Jasper Brown.. *RetinaNet Tutorial.* Retrieved May 22, 2022, from https://github.com/jaspereb/Retinanet-Tutorial

Assignment 3: Predicting on Streamed Textual Data

1. Introduction

The third assignment consisted of constructing a predictive model using Spark (Structured) Streaming and textual data. The data that is going to be analyzed comes from Twitch, more specifically chat messages posted on live channels.

The data gathered will come from two different channels: **loltyler1** (https://www.twitch.tv/loltyler1) (streams related to League of Legends game) and **asmongold** (https://www.twitch.tv/asmongold) (streams commenting news. On the day the comments were fetched, it was about Johnny Depp and Amber Heard trial). The channels were chosen because they stream about two different types of content, so the vocabulary and language used will be significantly distinct. Therefore, the model must be able to predict the channel based on a chat message and to make predictions in a "deployed" setting on new messages.

2. Constructing the data set using the provided stream

In order to have access to the chats, a **client ID** and a **Client secret** were required. We need to get this from the Twitch developer portal by registering our application. For connection with Twitch, there are three methods: by request, by socket and by twitchio API. The method used for this assignment was the last one. We need to obtain the token for the chat access from the link (https://twitchapps.com/tmi/) following the registration of the application in the portal. Following the keys for **token**, **client ID** and **client secret**, the Internet Relay Chat (IRC) was ready to access the chat logs.

Next, twitchio API and PySpark were connected by importing commands from **twitchio.ext**, **pandas** as **pd** and **SparkSession** from **pyspark.sql**. Additionally, **pyspark.ml.features** (**tokenizer**, **StopWordsRemover**, **CounterVectorizer**, **IDF**, **StringIndexer**) and **Pipepline** from **pyspark.ml** were imported to analyze and treat the text from the messages. Finally, to create a model, **LogisticRegression**, **GTBClassifier** and **LinearSVC** were imported from **pyspark.ml.classification**. The evaluation of the accuracy was done using the **MultiClassificationEvaluator**. Then the object **df** was specified to create a dataframe with defined columns. Next, a bot was initialized with the credentials and channel's names (for fetching chats) and a function was created in it to parse the messages and print **df** with the **author's** name, message **content**, **channel** name and message **date**. 10521 comments were fetched from channel **asmongold** and 3071 from channel **loltyler1**. After that, the bot was closed, and 3000 comments were sampled for each channel. Lastly, the comments were saved in a csv file, named *twitch-data2.csv*.

	author	content	channel	date
0	iwishiwasnotbored	GIGACHAD GET THIS BITCH	asmongold	2022-05-26
1	keyshock	She said to use the manual in her previous tes...	asmongold	2022-05-26
2	rikidybones	LOCK HER UP	asmongold	2022-05-26
3	savagepringle	He's cornering her	asmongold	2022-05-26
4	kimdracula93	REKT	asmongold	2022-05-26

Figure 26 - Table containing author, content, and channel of the messages

3. Constructing a predictive model to predict the channel based on the chat message

In this step, the object **df_spark** was created to load the csv file as a spark dataframe. Some preprocessing methods were applied, such as dropping duplicates and cleaning names channels if messy, resulting in 2938 comments from **asmongold** and 2835 from **loltyler1**.

Then the text mining analysis began with the comments being transformed into tokens and the stop words being removed from them. Then they were vectorized and the inverse data frequency (IDF) was applied. Next each channel was labeled (being 0 the label for channel **asmongold** and 1 for channel **loltyler1**) and applied to the dataframe.

```
+------------------+--------------------+---------+----------+-----+
|            author|             content|  channel|      date|label|
+------------------+--------------------+---------+----------+-----+
|infectiousinterest|Orient yourself t...|asmongold|2022-05-26|  0.0|
|  liquidcaliber484|need to bring up ...|asmongold|2022-05-26|  0.0|
|          itsbimaa|Madge ppOverheat ...|asmongold|2022-05-26|  0.0|
|       rabbit_xxxx|"half her money c...|asmongold|2022-05-26|  0.0|
|    akshan_gaming1|          WAYTOODANK|loltyler1|2022-05-26|  1.0|
+------------------+--------------------+---------+----------+-----+
only showing top 5 rows
```

Figure 27 – Labeling the channels

The data frame was randomly divided into 70% training and 30% testing. After that, three models were used to see which one would get the highest accuracy: Logistic Regression, Gradient Boosting Tree and Linear Support Vector Machine (LSVM). The method was the same for all the models: the pipeline was set to fit the model and then it was executed to make predictions in the test set, but only in the filtered columns (**rawPrediction**, **channel**, **label**, **prediction**).

Logistic Regression presented an accuracy of 81%, Gradient Boosting Tree of 68.8% and LSVM of 80.4%. Even though LSVM gives a lower accuracy than Logistic Regression, it was chosen because it can find good parameter settings automatically, being a very easy-to-use method for learning text classifiers. The LSVM model was saved in the *twitch.model* file.

4. Applying the trained model to predict as the stream comes in

First, the model was loaded, and a counter was created to see how many predictions it was able to do correctly. Then, the same process to fetch streamed comments was executed again: a bot was initialized to stream the selected channels and the same data frame was used

to parse the messages. Additionally, the dictionary previously conceived was converted to dataframe pandas and that was converted to dataframe spark. Next the labeled values were predicted using a single model for spark dataframe and only for the filtered columns.

```
Correct count: 174 & Wrong count: 90
######### 264 ##########
+--------+--------------------+---------+-----+----------+
|  author|             content|  channel|label|prediction|
+--------+--------------------+---------+-----+----------+
|fossabot|@xTrufiane, ACCOU...|loltyler1|  1.0|       1.0|
+--------+--------------------+---------+-----+----------+

Correct count: 175 & Wrong count: 90
######### 265 ##########
+------------+-------+---------+-----+----------+
|      author|content|  channel|label|prediction|
+------------+-------+---------+-----+----------+
|returning_anm|  Shush|asmongold|  0.0|       1.0|
+------------+-------+---------+-----+----------+

Correct count: 175 & Wrong count: 91
######### 266 ##########
```

Figure 28 – Predicting the channel based on chat message

5. Conclusion

Apache spark is known for its speed for data mining and processing for big data. It supports the common libraries such as Pandas and Numpy. In this assignment we applied the multiclass text classification using Pyspark. The received data from twitch mining was processed and prepared for classification models. After feature engineering, we designed the machine learning pipeline to apply the classification models. We applied 3 classification model types, and we decided to stick with Linear SVC for predicting real time chats.

To set up the code, the reader just needs to add the D0S07a.ipynb to the notebook folder in the spark directory provided for the assignment along with the twitch-data2.csv. Once the files are added to the folder and the spark is initiated, the reader can easily run the code and analyze the twitch data mining, pyspark classification models and real time classification from Twitch streaming.

6. References

(n.d.). *Twitch Chat & Chatbots*. Retrieved May 8, from https://dev.twitch.tv/docs/irc

(n.d.) (2021). *Multi-Class Text Classification with PySpark*. Retrieved May8, from https://www.section.io/engineering-education/multiclass-text-classification-with-pyspark/

Assignment 4: Exploring Graphs with Neo4j and Gephi

1. Introduction

In the fourth assignment we explored graphs with Neo4j and Gephi. Graph data was provided by the teaching staff and it included video and user data from the vastly spread social network TikTok. At the core of it were videos with tags related to the conflict in Ukraine, however the recommendations algorithm quickly branched out to various other topics.

We used the Neo4j desktop version to get acquainted with the software and do some initial exploration of the data, however we found it easier to write and share the code using Jupyter Notebooks and their extension for Neo4j. Some additional visualizations were then done in another software new to us, that is Gephi.

2. The Background

TikTok is a massive social media app. It started off by a Chinese company acquiring Musical.ly, an app which was at the time quite popular, with some hundred million downloads. Then the lift-off truly began and has not reached its peak yet, with the app's current download count at 1.6 billion.

Such a mass of people has a lot of power and TikTok has influenced many aspects of life. From music artists complaining they are forced by their record labels to post viral videos on the platform, pornographic content, and dangerous "challenges" (young) people do, to of course, the conflict in Ukraine.

This has been branded as the first "TikTok war", as the platform has been used to share videos from the fighting areas, propaganda and so on. Videos from the battlefield have emerged all over the internet and have received large amounts of views on both sides of the conflict. Such is the impact of the social network that President Biden even met with a few most influential users to discuss the way to share content. Numerous fake videos have sprung out as well, with edits making some old videos look problematic or just clips of other battlefields being tagged as related to Ukraine or Russia.

Shortly after the conflict began, many of the largest social networks, among them Facebook, Twitter and Instagram, shut down their services in Russia. This was not the case for TikTok. Their platform remained available, but then the ominous "fake news" law has been passed in the Duma, threatening prison sentences of up to 15 years to people accused of spreading false information – what that was, is of course up to the prosecutors to decide. In order to protect their users, TikTok then disabled uploading videos in Russia – but not watching them. The network remained available, but just without current Russian content. This is where things get interesting.

A newspaper report, published in early April, created two accounts on the platform – one with an IP address in Ukrainian Kharkov, and the other one just across the border in Belgorod; both users were 19 year old white men with AI generated profile pictures. After some videos "watched" by both bots it quickly became apparent that the content that Russian users are being showed differs massively from what Ukrainian users see – in Russia, virtually no clips of the war was shown, even though it was geographically very close, while conflict related videos were filling up the screen of the Ukraine-based account. After 4000 videos watched, only 13 were seen by both. Even after specifically searching for war related clips by the Russian account, none appear – they simply are not available there.

With that in mind, we started off our exploration of the data. While the results we found might not be as spectacular as what the previously mentioned report discovered, it was still interesting to dive into the data of one of the most controversial and popular social media apps of our time.

3. The Work

The initial part of our journey was dedicated to getting familiar with Cypher, the query language used in Neo4j, and the data we had available. As the members of the group focused on this assignment were not active TikTok users, we wanted to gain some initial feeling for the impact the videos had. The most watched video in our dataset had almost 700 million views, has been liked 210 million times and was made by an account with almost 90 million followers. If we only consider videos that mention Ukraine or Russia in their description, the most played video has over 50 million views and the fifth most almost 40 million – all of them in support of Ukraine. However, none of those were published by accounts with "news" in their ID.

3.1. Recommendations

From the beginning we were interested in the way videos and users get recommended to each other, as we deemed this very important to the way content was spread – users only get recommended other users, and from videos we get other videos.

As we have seen in the introduction, the recommendations algorithm works in mysterious ways. Most recommended video got recommended 14 times but was nowhere near the most played one – it got less than 2 million views. None of the top 25 most recommended videos mention the war in their description.

Accounts get recommended more; however, we believe this is because of the nature of the app – when on your profile, multiple recommended accounts to follow appear. The profile with most recommendations was suggested 151 times. In the top 25 most recommended accounts (all had at

least 138 recommendations) there are multiple Belgian accounts – most likely because data was scraped in Belgium.

Interestingly, users of most recommended videos were seldom suggested more than once, with one very clear outlier being recommended 149 times – which is a rather big Belgian TikToker. This user also had 18 of their videos recommended a total of 44 times – both of these numbers are the highest in our dataset, which can once again be attributed to the geographical location. As for centrality measures, the highest one belonged to news related accounts – which can most likely be attributed to the way the data was gathered, as those accounts published a lot of Ukraine and Russia related videos.

3.2. Weakly connected components

We continue our investigation of recommendations. Firstly, we create two projections of our original graph – in one, we store data on users and recommendations between them, and the same for videos in another. This is done to be so we could freely meddle with the data without the fear of contaminating the original dataset, while also making the computations faster, since the projected graph is smaller and has fewer connections.

3.2.1. User recommendation

Let us closely look at the way users are connected between each other via recommendations. This is a directed graph – edges point from the user to whom the recommendation was made to the recommended account. We search for *weakly connected components;* in a directed graph a WCC is a subgraph where all vertices are connected by some path, ignoring the direction of edges.

We instantly see that most of the data lies in one very large component with 109.561 vertices; the second one only contains 617, while the third 330. The number quickly declines after that. There is an unusual trend in the size of WCCs – there is 77 components sized 31, 40 sized 30, and 15 components sized 29 – after that, all component sizes are present 8 times or less. These components sized 29-31 also have a similar feature in that there is one central component, to which all the others were recommended – so between 31 vertices there are 30 connections, all going from one node to all the others, as seen in figure 29. The size of the largest component where Ukraine or Russia isn't mentioned in any of the user's info is 330.

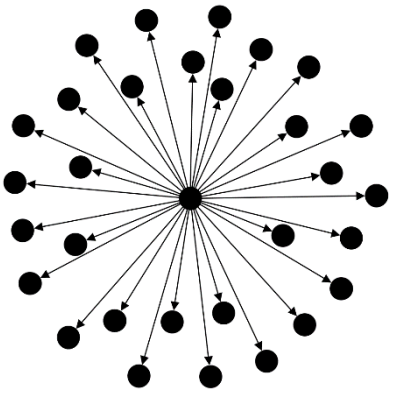

Figure 29 - example of a WCC with 31 nodes and 30 connections, all from one central vertex to the others

3.2.2. Video recommendations

Similar was done for the projected graph containing only videos and recommendations between them. As mentioned earlier, there are fewer relationships between videos, as the most recommended one received 14 recommendations – as opposed to 151 with users. As was the case with the users, also now we found one very large WCC with 22800 videos, while the next one was of size 7.

Because of the nature of the recommendations process regarding the videos, we realized many are potentially not connected – they were found on a user's site, but not via recommendations. Our suspicions were correct, since we found 114474 videos which were not recommended. There does not seem to be any correlation with the video being recommended or it mentioning Ukraine or Russia. Most non-recommended videos per account was 93 (out of 2640 videos made by this user), once again from a (different) Belgian TikToker – none of those videos mentioned the conflict in Eastern Europe. However, 50% of videos uploaded by account »visitukraine_« were not recommended, by far the largest percentage (the next one had about 20% of their videos disconnected).

3.2.3. Random walks

As the last part of our project, we looked into random walks between videos and users. We searched for a node with a tag, song or description related to Ukraine or Russia and randomly selected videos related to it.

When dealing with videos, looking at any other but the largest WCC is certainly pointless (since they contain at most 7 nodes); the length of the longest walk we could find, starting from a connected node mentioning Ukraine or Russia to a node where there is no way forward is 8. Another aspect we looked into is how long does it take, starting from a Ukraine-or-Russia tagged video, to reach a video not containing those tags – in most cases, the next recommended video

was without them already. However, looking at walks where at least two consecutive (initial one and then another one) videos mention the conflict gives some additional aspect. We see some communities of Ukraine/Russia related videos form, but due to the direction of the connection, no path can be found there. This is visible in figure 30, where blue nodes indicate that the video mentioned Russia or Ukraine, and the pink colour shows the lack of those tags. The node gets depicted bigger, if it has a higher in-degree – it has been recommended more.

The same was then done with users as well. The starting nodes for the random walks were users who mention Ukraine or Russia in their userinfo or whose videos were related to Ukraine or Russia in the manner described before. Since the number of users to start a walk from was a lot higher than the number of videos, a random sample was taken as the source nodes. For the users, we increased the number of consecutive components mentioning the conflict to three (initial one and two more consecutively). The large group at the center of figure 31 contains most of the news accounts. Centrality measures were also calculated for this group: in- and out-degree, two types of closeness, betweenness, and PageRank. For all these centralities, news accounts were often found to have higher scores.

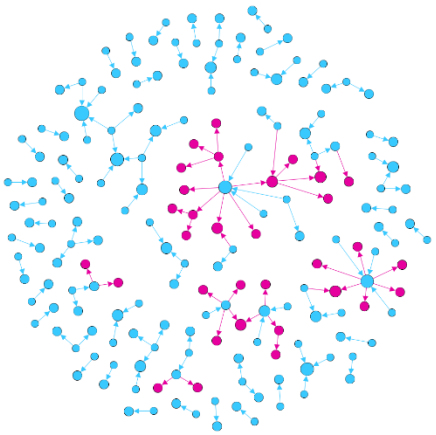

Figure 30: random walks of videos with at least two consecutive mentions of Ukraine or Russia at the start of the walk

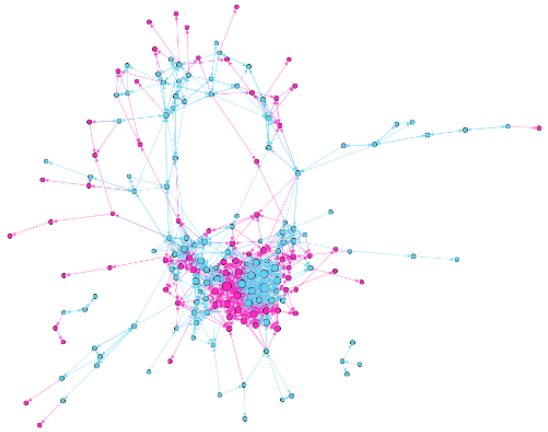

Figure 31: random walks of users with at least three consecutive mentions of Ukraine or Russia at the start of the walk

4. Conclusion

As was the case with Ukrainian and Russian user's feed and recommendations mentioned in the introduction, we also notice a trend based on the data gathered in Belgium, but the situation is not as dire as in Russia (thankfully). Recommendations also do not necessarily suggest most viewed videos or videos from most followed accounts, but must be based on some other, most likely more elaborate measurement. In the last part we also see that not many videos about Ukraine or Russia are present in a network of videos gathered, even though data was gathered with them at the center – certainly there are not as many uploaded on the platform as other types of clips, but the lack of them is surprising.

Appendix

```
target                                1.000000
homebanking_active                    0.102598
bal_mortgage_loan                     0.102502
has_homebanking                       0.099479
has_mortgage_loan                     0.099355
has_life_insurance_decreasing_cap     0.098371
has_current_account                   0.094351
cap_life_insurance_decreasing_cap     0.087994
bal_savings_account                  -0.069249
has_personal_loan                     0.068168
bal_current_account                   0.066677
customer_age                         -0.060932
bal_personal_loan                     0.060565
has_savings_account                  -0.046596
visits_distinct_so                    0.041338
bal_current_account_starter           0.037614
customer_gender                      -0.029663
has_current_account_starter           0.028129
visits_distinct_so_areas              0.024931
has_fire_car_other_insurance          0.024872
prem_fire_car_other_insurance         0.024512
has_savings_account_starter           0.021084
bal_savings_account_starter           0.017540
has_insurance_21                      0.016326
bal_insurance_21                      0.013291
customer_self_employed                0.011613
has_insurance_23                      0.009326
bal_insurance_23                      0.007824
has_pension_saving                    0.006929
has_life_insurance_fixed_cap         -0.006341
cap_life_insurance_fixed_cap         -0.005168
bal_pension_saving                   -0.002085
Name: target, dtype: float64
```

Appendix 1 – Correlation of the variables with feature Target