KU LEUVEN

FACULTY OF SCIENCE

Advanced Analytics in Business
Lab Report

Advanced Analytics in Business
Prof Seppe vanden Broucke
Sunday 30 May 2021

Group 9

Nicolas Arturo Cozzi Elzo	r0820077
Andreas De Ryck	r0706846
Katherine Anne Sarapata	r0774860
Marie Analiz April Limpoco	r0820249
Khachatur Papikyan	r0825613
Benjamin Fabian Hucke	r0779535

Contents

ASSIGNMENT 1: PREDICTIVE MODELING ON TABULAR DATA

The task in assignment 1 was to construct a model that successfully predicts fraud based on a data of a car insurance company in Belgium. A training and a test set where given. The training set contained 78 features including the two dependent variables *fraud* and *claim_amount* and 55,463 instances, whilst the test set did not include the dependent variables. The modelling approach was to split the training set into two subsets, *subtrain.train* (80% of instances) and *subtrain.test* (20% of instances) and fit a model on the *subtrain.train*. The model performance was then evaluated on the *subtrain.test* before applying it the actual test set of 2018. Obtaining a quite large training set with 80% of the observations in the original dataset was considered essential to capture relevant patterns in the highly unbalanced dataset. A stratified splitting of data was not deemed to be necessary because the frequency of fraud cases was nearly identical after the split. RStudio was used to code.

Pre-processing and data quality issues

In a first data cleaning step, commas in the *claim_amount* variable were replaced by dots and the variable was recoded to a numeric one. In a first exploration of the dependent variables, it is observed that high claim amounts are likely to be fraudulent claims, which should be kept in mind for the pre-processing steps since it shows that the variables are related to each other.

After that, steps were carried out to gain an overview over missing values in the data set. These steps included counting missing values per feature and instance. Here, it emerges that multiple features do not contain any or only very limited data as they have close to or 100% missing values. Those are almost entirely variables about a potential third parties in a claim. However, at this stage no variables were excluded from the data yet. The percentage of missing values per instance was added to the data frames as a new feature as it might be meaningful to predict fraud cases.

In further pre-processing steps, Boolean and character variables are set to factor variables, missing values are provisionally set to -999 for numeric variables and to ‘unknown’ for factor variables and implausible birth dates are set to missing.

Missing values

The general approach chosen for dealing with missing values was to impute them whenever possible. For numerical variables missing values were replaced by the median of the variable in the training set in case the number of missing values was not considered to be too high. For factor variables, missing values were set to 'unknown', which was introduced as a regular factor level in the variables. The reason for that was that missing values might potentially not occur randomly, but instead be meaningful for predicting fraud cases. Hence, variables such as *third_party_1_expert_id* that have 99% missing values were kept in the dataset with missing values just being set to 'unknown'. Only features with 100% missing values and the numeric variable *claim_time_occured* (88% missing values) are deleted from the dataset. The feature *claim_time_occurred* is deleted because no valid imputation of missing values was found. An alternative would have been to recode the feature to a factor and introducing 'unknown' as a factor level to replace missing values.

Modelling strategy

Different approaches for the modelling strategy were considered. Firstly, the issue of unbalanced data should be addressed, as less than 1% of the instances were actual fraud cases. Therefore, undersampling and oversampling were discussed as techniques to deal with the unbalanced dataset. However, none of these techniques were found to improve the model fit.

Our modeling goal was to favor specificity over sensitivity with special attention devoted to fraud cases with a high claim amount. The reasons for that were that a high sensitivity would come along with a higher number of false-positives which we concluded was not desirable for a car insurance. Instead, we concluded that favoring specificity in general, but aiming for a high sensitivity among frauds with a high claim amount would give the car insurance the chance for effective and feasible investigations of suspicious cases. This would come at the danger of not detecting frauds with small claim amounts, which might be considered negligible by the insurance in case they do not exceed an unreasonably high amount.

Featurization

Featurization steps included manipulating existing features in order to make model construction more feasible, and creating new features based on information from at least two other features. First featurization steps are carried out on all features containing postal codes. In order to obtain postal code factor variables with a reasonable amount of factor levels, only the first digit of each postal code is kept in the data. In Belgium, the first digit corresponds to a certain region, which makes it a useful indicator for the region a driver is from or where the accident happened. A bivariate exploration of the relationship between the policy holder postal codes and fraud shows that fraud cases are more frequent if policy holders are based in municipalities whose postal codes starts with 5, 6 or 7, which corresponds to the provinces of Namur, Hainaut and Luxembourg. Hence, featurized postal codes are considered potentially relevant variables. Furthermore, years, months, and days are extracted from all features containing dates. Other newly created features are shown in Table 1.

Table 1. Newly created features

Feature Name	**Description**
days_bet_acc_claim	Days between accident and claim
claim_wday_reg	Weekday the claim was registered
yrs_policy_start_acc	Years between policy date start and accident
days_acc_next_expiry	Days between day of accident and next expiry of the policy
days_pol_start_next_expiry	Days between policy start and next expiry
yrs_vehicle_inuse_acc	Years between date the vehicle came into use and accident
diff_veh_3rd_party	Difference between number of vehicles involved in the claim and number of third parties
diff_veh_injured	Difference between number of vehicles involved and number of injured individuals
diff_3rd_party_injured	Difference between number of third parties and number of injured individuals
same_expert_ph_dr	Is the expert ID of the policy holder the same as the expert ID of the driver? (Yes or No)

same_expert_3rdpy1_ph	Is the expert ID of the policy holder the same as the expert ID of the third party involved? (Yes or No)
same_expert_3rdpy2_ph	Is the expert ID of the policy holder the same as the expert ID of the second third party involved? (Yes or No)
same_expert_3rdpy1_3rdpy2	Is the expert ID of the first third party the same as the expert ID of the second third party involved? (Yes or No)
same_postal_code_ph_dr	Do the policy holder and the driver have the same (full) postal code? (Yes or No)
same_postal_code_ph_3rdpy	Do the policy holder and the third party involved have the same (full) postal code? (Yes or No)
same_postal_code_dr_3rdpy	Do the driver and the third party involved have the same (full) postal code? (Yes or No)
same_vehicle_ID_dr_claim	Are the claim vehicle ID and the driver vehicle ID identical? (Yes or No)

In the next part of the featurization, all variables containing IDs were dealt with. These can be split up into 4 different families: Repair IDs, expert IDs, Person IDs, and vehicle IDs.

The goal was to create a list of suspicious IDs for each of the 4 families. In terms of the person IDs for instance, this would mean that an ID that was involved in fraud as a policy holder **or** as a driver **or** as a third party is always considered suspicious, even if he has only been e.g. involved in a fraud as a third party. By handling IDs like this, it is ensured that suspicious IDs are found as such even if they tend to be involved in frauds only in one of the sub-variables within a family. In case an ID was included into the list of suspicious IDs, they were kept in the dataset, whilst unsuspicious IDs were labeled as '*Not suspicious'*. The same procedure was applied for each ID variable. All IDs present in each ID variable were evaluated in terms of how often they appear in one variable and how often they were involved in fraud cases. This way, the total number of frauds and the percent of fraud cases within all cases a certain ID was present were obtained.

The decision as to when an ID is considered suspicious was made separately for each of the four ID families based on theoretical considerations. For efficient featurization steps and the way of identifying suspicious IDs, the modeling strategy should be clear already. Hence, it is

important to think about goals of the model that were specified above because of the effects of featurization on specificity or sensitivity. Since our approach was to favor specificity, we tended to set higher requirements for IDs to be considered suspicious.

For suspicious repair IDs, every repair shop that was involved in fraud at least once was added to the list of suspicious IDs as it was assumed that repair shops are potentially powerful accomplices in fraud because of being seemingly neutral. 112 repair IDs were found to having been involved in fraud. Those IDs were subsequently kept in the data, whilst others were set to '*not suspicious*'. While checking suspicious repair IDs, one extremely suspicious shop was found which was involved in fraud 13 out of the 15 times it appears in the data. Hence, it was decided to add an additional variable that specifically labeled only this repair shop as '*suspicious'* and set all other to '*not suspicious'*.

For expert IDs, a slightly different approach was chosen. Considering the large number of experts who worked for the drivers, policy holders or third parties in fraud cases, it was chosen that only experts for whom the percentage of fraud cases among all cases they appeared in was higher than 10% percent were included in the list. Additionally, taking the correlation between *fraud* and *claim_amount* into account, it was decided that experts that were involved in at least one fraud case that exceeded 10,000 euros are also added to the list of suspicious experts. This allows to filter out experts that might be involved in frauds with a high claim amount even if they do not frequently appear in fraud cases. That way, 26 suspicious experts are filtered whose IDs are kept in the data set, whilst all others are set to 'Not suspicious'. Similarly to suspicious repair IDs, an additional dummy variable with levels '*Suspicious'* and '*Not suspicious'* is added to account for very suspicious experts. Those are experts that were involved in fraud cases at least 5 times and whose percent of fraud cases among all cases they appear in is at least 50%. They were labeled as 'Suspicious' in the additional variable.

Next, suspicious person IDs were investigated. The same procedure was applied as before, and a list of suspicious IDs was obtained. Policy holder, driver or third party IDs were considered suspicious if there were at least two fraud cases associated with them in the training set. This value was chosen because appearing in a fraud case once is always possible even without being involved in the actual fraud. Thus, choosing this value is in line with the goal of a satisfactory specificity.

For suspicious vehicle IDs, only vehicles that appear twice in connection to a fraud case within the sub-variables claim vehicle, driver vehicle, or third party vehicles were considered

suspicious. However, no vehicle appearing twice within one variable was found. Hence, no vehicles were considered suspicious.

Table 2. Newly created features dealing with suspicious IDs

Feature Name	**Description**
Susp_repair_id	Distinguishing between repair shops that appear in fraud cases (suspicious) and those who do not (unsuspicious)
Susp1_repair_id	Distinguishing between very suspicious shops (involved in more than 5 frauds and more than 10% fraud, marked as suspicious) and others
susp_policy_holder_expert_id	Variable containing the IDs of suspicious policy holder experts, others are set to 'not suspicious'
susp_third_party_1_expert_id	Variable containing the IDs of suspicious third party experts, others are set to 'not suspicious'
driver_expert_id	Variable containing the IDs of suspicious driver experts, others are set to 'not suspicious'
susp_policy_holder_id	Variable containing the IDs of suspicious policy holders, others are set to 'not suspicious'
susp_third_party_1_id	Variable containing the IDs of suspicious third parties, others are set to 'not suspicious'
susp_driver_id	Variable containing the IDs of suspicious drivers, others are set to 'not suspicious'
susp1_driver_expert_id	Distinguishing between very suspicious drivers (involved in more than 5 frauds and more than 10% fraud, marked as suspicious) and others
susp1_ policy_holder_id	Distinguishing between very suspicious policy holders (involved in more than 5 frauds and more than 10% fraud, marked as suspicious) and others

The processes of identifying suspicious IDs should be critically reflected in retrospective, as the chosen method probably left some suspicious IDs undetected. The chosen approach was to look for suspicious IDs **within** sub-variables of the different ID families and then generalize suspicious IDs on other sub-variables of the same family. Probably, a better approach would have been to search for suspicious IDs **over** the sub-variables of the same family from the beginning. For instance, based on the chosen approach, a hypothetical person ID that appears only twice in the whole dataset, once as policy holder ID in a fraud case and once as a driver ID in a fraud case would not be detected. Similarly, regarding vehicle IDs, investigating IDs over sub-variables would have led to the detection of some suspicious vehicles. Thus, changing the featurization of suspicious IDs likely would have contributed to more accurate predictions of fraud.

Results and reflections on model performance

As a supervised learning technique, random forest with *fraud* as dependent variable was chosen as the technique to be used. The reason for that was the assumed non-linearity of the decision boundary as well as the assumption of interaction effects between features. For the model, 10,000 trees were grown with eight variables considered at each split of the tree. Additionally, a linear prediction of *claim_amount* based on all independent variables was considered to be added to random forest model trying to predict fraud. However, it proved difficult to establish reliable predictions of *claim_amount* based on the independent variables in the dataset. Hence, a prediction of *claim_amount* is not included in the random forest model, which was not deemed to be a big downside because *claim_amount* was already accounted for during the featurization steps by also finding suspicious IDs based on high *claim_amount*.

Applying the random forest model on the test set of 2017 showed that predictions are satisfactory. Choosing a cut-off value of around 0.95, 36 out of 56 fraud cases were correctly classified (see Table 3.1). The number of false-positives lies as 210, which is roughly 2% of all negative cases. However, for frauds with a high claim amount (> 8,000) the sensitivity equals 100%, whilst the false-positive rate is around 8%. In a real business setting, the chosen cutoff might have to be adjusted depending on the available resources and costs of further investigating potential frauds.

Regardless of this, the results indicate that, overall, the model is capable of successfully predicting fraud cases, especially those with a high claim amount, while keeping the false-positive rate reasonably low. Still, from the practical standpoint of fraud detection, it would be highly

desirable to have a lower false-positive rate, which limits the utility of this model in practice even though it would have a noticeable contribution in the field of fraud detection.

Table 3. Classification Tables

Table 3.1. All cases

	Predicted: no fraud	*Predicted: fraud*
No fraud in reality	*10827*	*210*
Fraud in reality	*20*	*36*

Table 3.2. Cases with claim_amount > 8,000

	Predicted: no fraud	*Predicted: fraud*
No fraud in reality	*195*	*18*
Fraud in reality	*0*	*14*

Reflecting on the scores on the public leaderboard, the high score and the 5th highest recovered claim amount confirm that the model was successful in detecting fraud with high claim amounts. The AUC of the model which is probably only slightly above average (in comparison to other groups) indicates that the goal of a high specificity was not entirely met and could have been improved. When investigating false-positives in the data, it is likely that the newly created features of suspicious IDs had an impact. Only keeping IDs that were classified as suspicious and setting not suspicious ones to 'Not suspicious', has likely caused cases where a suspicious ID was involved to be classified as positive (or to be having a higher fraud probability). Increasing the thresholds for IDs for being considered suspicious would have likely led to a higher AUC, but to lower recovered funds, whereas decreasing them could have increased recoveries and decreased the AUC. Probably, the model would have benefited from additional fine tuning on that matter (possible approaches were described in the featurization paragraph). Still, the high amount of recoveries together with a reasonably high AUC form a satisfactory result on the public leaderboard.

Comparing the public score to the hidden score yields ambiguous conclusions. Firstly, both the score and the AUC improved on the hidden leaderboard, which gives further assurance that the model works well on unknown data. Looking at group comparisons of the differences between the public and hidden leaderboard, it is apparent that the recoveries increased for almost every group, whilst the AUC increased only for most, but not as many groups. Hence, the hidden 50% appear to include higher claim amounts in the top 100 cases. Because of the increasing

AUC for most groups, it also seems likely that fraud cases follow similar patterns as in the training set. It is noticeable that for some groups (such as group 1 or group 7) the recoveries increased very strongly and stronger than for our model, which indicates that multiple cases with a quite large claim amount were most likely detected by these groups and went undetected for us. By looking at the variable importance, it becomes clear that the backbone of successful fraud detection of high claim amounts is the appearance of certain suspicious IDs, most notably policy holder expert IDs or driver expert IDs (see Table 4).

Table 4. Variable importance in prediction

Variable Name	*Importance*
susp1_driver_expert_id	*61.6*
susp1_policy_holder_expert_id	*33.5*
susp_repair_id	*20.6*
susp_policy_holder_expert_id	*12.7*
susp_driver_expert_id	*12.1*
claim_cause	*7.8*
days_acc_next_expiry	*7.6*

Within this context, the predictions on the hidden dataset should be reflected. Doing that, it becomes clear that our model struggled to detect fraud cases with no suspicious IDs or very limited suspicious IDs involved. Some of those cases were probably identified by models of other groups and depend on features that we failed to establish. The fact that the difference between the scores is quite high between a few groups and the rest on the hidden leaderboard gives some indication that a fair number of fraud cases followed that pattern that we were not able to find. Investigating fraud cases in the hidden data where the predicted fraud probability of our model was below 15%, shows that our model had problems in identifying fraud when the claim cause was a traffic accident and no suspicious IDs were involved. For further model fine-tuning, these instances could be investigated to find possible features or indicators of frauds within these instances.

Reflection on measure used in the leaderboard

To determine whether the chosen measures in the leaderboard are best fitted to evaluate the quality of the model depends on what perspective one has in evaluating the model. From a functional standpoint, the use of recovered claim amounts from fraud cases carries a lot of weight for a car insurance that of course aims to recover as much as possible. Also, a higher score is without a doubt indicating a better working model on average. It is, however, also a slightly shortsighted way of evaluating the quality of a model and the differences in the recovered amounts between the public and hidden leaderboard indicate why. These partly remarkable differences (e.g. +35% for group 99) reveal a high variance of the score, which is based on the data situation and which makes the score as a metric to evaluate a model's performance unreliable to some degree. Using our own model as an example, it would seem very good on a leaderboard if by chance there were 4 more high claim amount frauds with suspicious policy holder IDs in a hypothetical new dataset that our model successfully predicts. Hence, applying the model on a new dataset would likely result in moderate changes in the recoveries that depend on the number of cases that have a certain pattern that one model successfully predicts. We might learn more about the reliability of the used score in a sample of multiple datasets that the model would be applied on. That way meaningful averages could be computed to evaluate models based on the amount of recoveries. It should be noted however, that the aforementioned dataset-based variance of scores only explains a part of the differences between the scores of different groups. Thus, the score can be regarded as a metric that helps in evaluating models. One should, however, be aware of its limitations.

Any alternatives to the score have different limitations as well. For example, just asking for a classification and sort models by precision and recall does not take the different claim amounts into account and would thus not favor correct classification of fraud cases with a high claim amounts. Hence, checking precision and recall for all instances might give a better indication of how good a model is, whilst it would be less useful in the setting of an insurance company.

In conclusion, the chosen measure in the leaderboard is probably the most appropriate from a functional point of view, as it is the best indicator of a model's usefulness for insurance companies. In an academic context, a consideration of precision and recall across the whole dataset might be more meaningful and reliable in terms of evaluating model performance.

ASSIGNMENT 2: DEEP LEARNING ON IMAGES

Overview of the deep learning pipeline

Data

To complete the task of building a deep learning model that predicts the number of likes per hour given an image from Instagram, the first challenge was loading the large image dataset and finding a way of associating it with the target, which is the number of likes per hour. This step was not straightforward because of the unstructured nature of the data in contrast to the typical tabular dataset handled in Assignment 1. To accomplish this, a function was created to read the *.png* images from its folder path and then store the resulting object as an array, which is the data structure required in the modelling process. This array consists of RGB values of the images serving as features that are used to predict the target. One image was problematic and so only 11,695 are included in the analyses.

Aside from reading the image dataset, resizing was done in order to account for the limited capacity of the computer used to run the program. The image size was set to 128 because above this value memory issues were arising. Another important data preprocessing step was the normalization of the RGB values from having an original range between 0 and 255 to a range between 0 and 1. This was done because neural networks tend to perform poorly on unscaled data. Particularly, the network learns very slowly whenever the inputs are very large. Figure 1 presents samples of the resized and normalized images.

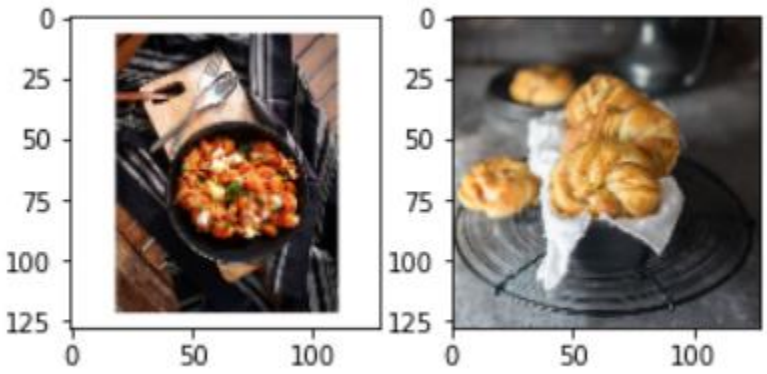
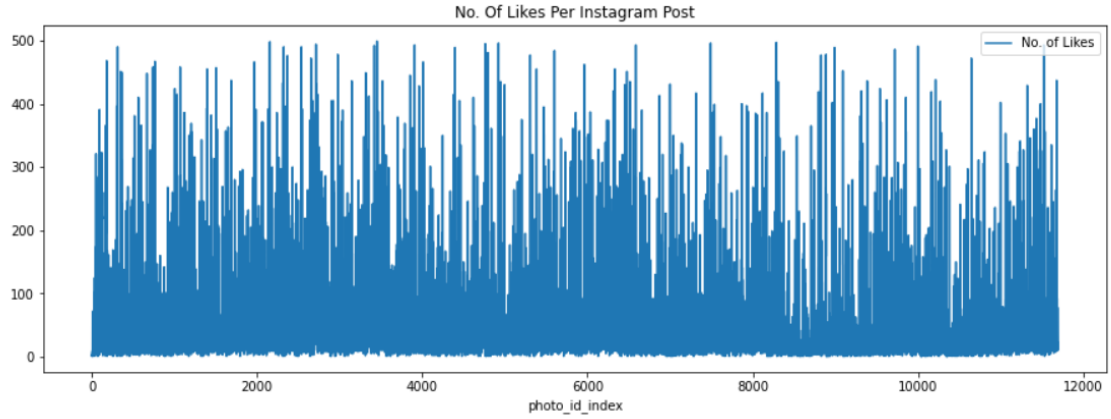
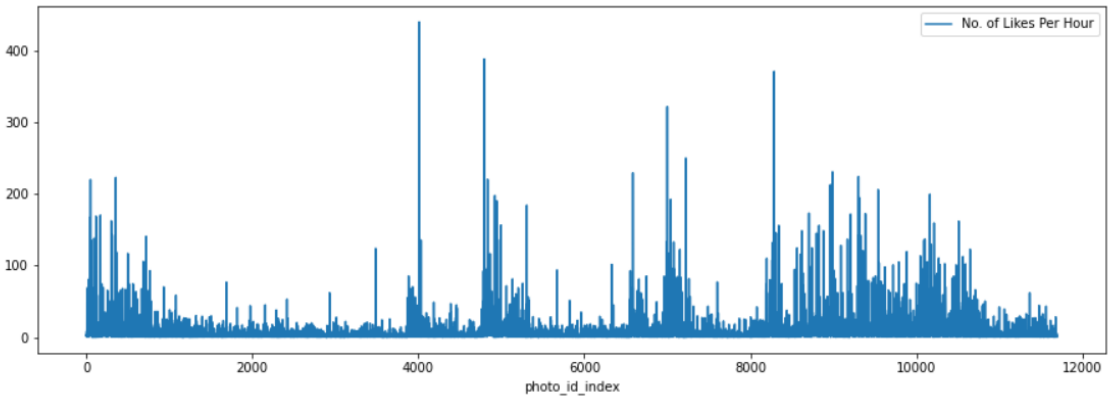


Figure 1. Sample of loaded, resized, and normalized Instagram photos

From the metadata, photo_id's, number of likes and the time difference between the scraping and posting of the image (in hours) were selected. The regression task focused on determining the number of likes *per hour* in order to have an equal scaling among the images (i.e. a photo might have many likes not because of the photo itself but because it had been posted earlier than another recently uploaded one). Figure 2 shows the plots for the number of likes and the number of likes per hour for the 11,695 photos.



(a)



(b)

Figure 2. The (a) number of likes and (b) number of likes per hour for the 11,695 images from Instagram

After putting the RGB numerical values of the image data and the number of likes per hour in an array, the dataset was split into training and test sets to be able to evaluate the generalization performance of the model later on. This test data will in no way influence the adjustments on the

model to improve its performance. To have an idea though on how to adjust the model parameters and hyperparameters to improve performance, 10-fold cross validation was conducted on the training set and the mean absolute error across all the folds was used as an estimate of the generalization ability of the model even before checking the model on the test data.

Model

Transfer learning is a technique of using a previously trained model on a particular dataset for another task similar to the goal of the previous task but with a different dataset. In this case, the weights from the existing best performing model trained on another dataset are used as initial weights for the task at hand. In this assignment, this technique was adopted to avoid problems associated with poor random initialization of the network's weights. Specifically, we used the *EfficientNetB0* as the base model with weights from ImageNet because it is one of the top-performing models with a relatively smaller size compared to the other transfer learning models (https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html). A sufficiently small network size helps avoid the danger of having too many parameters to train that may eventually lead to overfitting—a problem wherein the model memorizes the training data but has very poor generalization performance on unseen data.

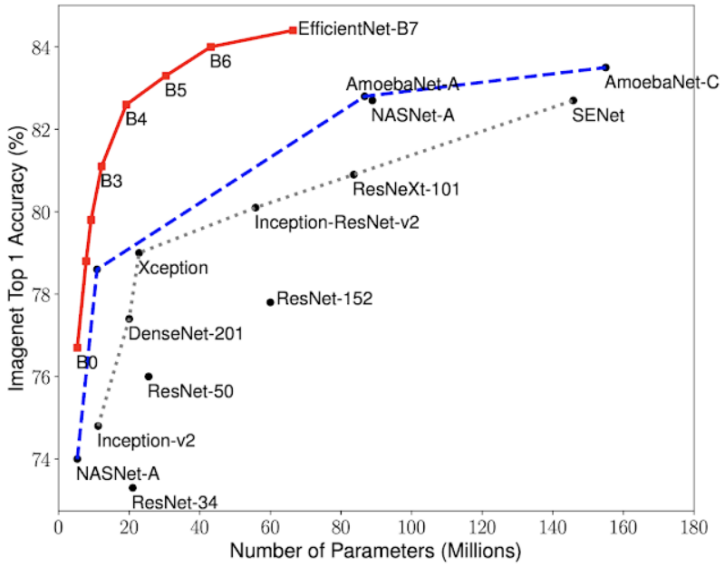


Figure 3. Comparison of CNN models in terms of accuracy and network size (https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html)

EfficientNetB0 has a total of 4,049,571 parameters and takes input images with a size of at least 224x224x3. As mentioned previously, the image size for this assignment was set to just 128x128x3 because of memory constraints. Additionally, *EfficientNetB0* was originally designed to perform a classification task. Hence, to achieve the present goal which is a regression task, the top layer of the base model was removed and adapted.

Layers that perform global average pooling, batch normalization, and dropout were added. The global average pooling is an operation which downscales the output from the previous layer by computing the average on a specified feature map size and using this average as an input to the next layer. Here, the global average pooling is done on two input channels. Dropout is then applied to help prevent overfitting since it literally randomly "drops" a portion of the units from a given layer. Finally, the output layer is a dense layer which means that all units from a previous layer are fully connected to the units of the succeeding layer, as opposed to convolutional layers.

Mean Squared Error (MSE) as the loss function, Mean Absolute Percentage Error (MAPE) and Root Mean Squared Error are chosen as the measures to assess performance on the validation set, which is 20% of the training set.

Hyperparameter tuning

Hyperparameter tuning while training a neural network involves setting the best possible values for the hyperparameters such as (but not limited to) the number of layers, number of hidden units, the type of layer to include, the optimization algorithm to use, and the learning rate. The neural network architecture has a crucial impact on the performance of the model not only on training data but most specifically on unseen data. Hence, it is important to tune the hyperparameters to obtain the best possible network. This is typically done using a 10-fold cross-validation wherein the training data is divided into 10 folds and for each value of the hyperparameter, the model is trained on 9 randomly selected folds and its performance is validated on the unused fold. The mean loss is then computed and serves as the performance measure of that particular hyperparameter value. The hyperparameter value that results in the best performance is then selected.

For the current case, a 10-fold cross-validation on the learning rate of the Adam optimizer was conducted. The Adam optimizer computes the adaptive learning rate for each parameter and is known to perform rather well in comparison with other optimizers (https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html#:~:text=Optimizers%20are%20algorithms%20or%20methods,problems%20by%20minimizing%20the%20function). However, its parameter which is the learning rate needs to be tuned well. Grid search was used to accomplish this, but the values used were not exhaustive due to time and memory constraints. MAPE served as the scoring metric for choosing the optimal learning rate because the other metrics such as RMSE, MSE, and MAE did not show significant changes from one value to the other and so the effect of changing the learning rate was not very pronounced.

Regularization

Avoiding overfitting is one of the challenging tasks when training a huge network. Different regularization techniques can be performed to address this potential problem. In this assignment, aside from implementing dropout in the network itself, which is an explicit means of regularizing, freezing of the weights from the pre-trained *EfficientNetB0* model was also done in order to limit the number of parameters to train. Figure 4 shows the summary of the model architecture wherein only 3,841 of the 4,055,972 parameters can be trained.

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_2 (Sequential)    (None, 128, 128, 3)       0
_________________________________________________________________
efficientnetb0 (Functional)  (None, 4, 4, 1280)        4049571
_________________________________________________________________
global_average_pooling2d_3 ( (None, 1280)              0
```

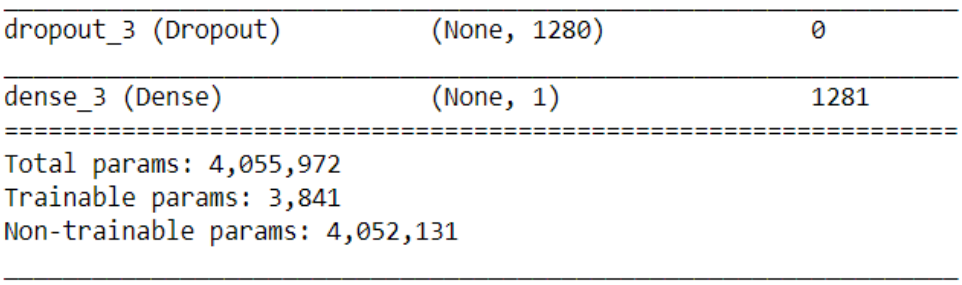


Figure 4. Model summary

Data augmentation was also implemented to provide variations of the images. This reduces the tendency of the network to memorize the training data and perform poorly on test data. Another form of regularization that was employed is early stopping. This was done by monitoring the loss *on the validation set* simultaneously with the training loss. As expected, the error on the training data decreases with the number of epochs. However, for the validation data, there comes a point beyond which the error starts to increase even though the training error still continues to decrease. This signals the end of training because beyond this point, the model will surely overfit. A maximum of 10 increasing validation error values were set to signal the stop of training. The validation set was obtained by splitting the training data further into 20% for validation and 80% for model construction.

Fine-tuning

This step is optional but has the potential to improve model performance. This step was done after attaining the lowest possible validation error. The weights of the base model were unfrozen and the entire network was trained such that all of the weights including those from the base model were updated according to a small learning rate (1e-5) to avoid overfitting. In this way, the initially generic weights from the base model were adapted to the specific task. The model summary of the fine-tuned model is shown in Figure 5 where the total number of parameters trained is 4,008,829 since the weights from the *EfficientNetB0* model were already included in the training and updating.

```
Model: "sequential_53"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_51 (Sequential)   (None, 128, 128, 3)       0
_________________________________________________________________
efficientnetb0 (Functional)  (None, 4, 4, 1280)        4049571
_________________________________________________________________
global_average_pooling2d_52  (None, 1280)              0
_________________________________________________________________
dropout_52 (Dropout)         (None, 1280)              0
_________________________________________________________________
dense_52 (Dense)             (None, 1)                 1281
=================================================================
Total params: 4,050,852
Trainable params: 4,008,829
Non-trainable params: 42,023
```

Figure 5. Model summary of the fine-tuned model

Results

After conducting 10-fold cross-validation over the possible values of the learning rate considered for the Adam optimizer, the optimal value recommended was 1e-5. The model was then trained using this learning rate value and the MSE was monitored for the validation data such that if there were no more improvements after *patience* = 10 epochs, then training would stop. The results are shown in Figure 6.

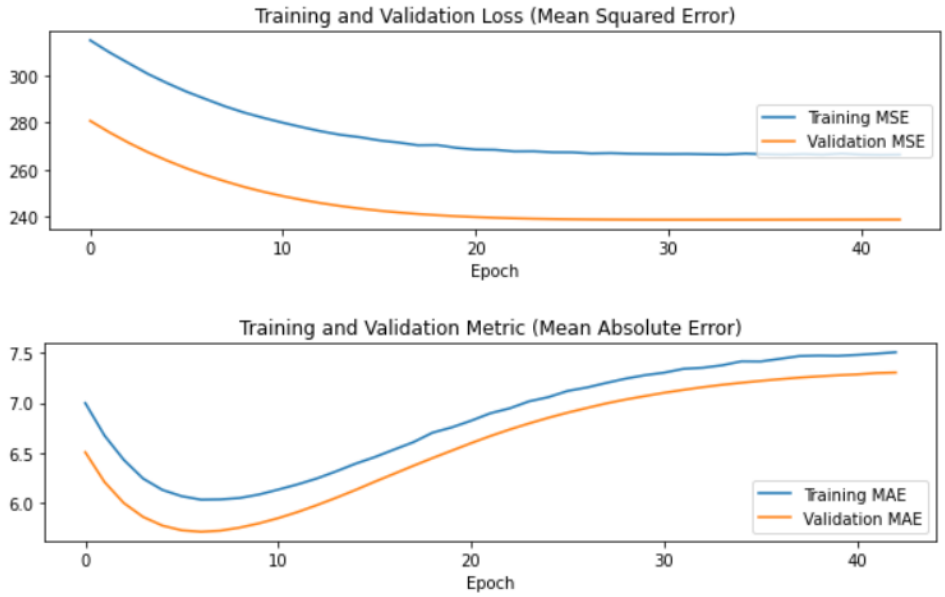
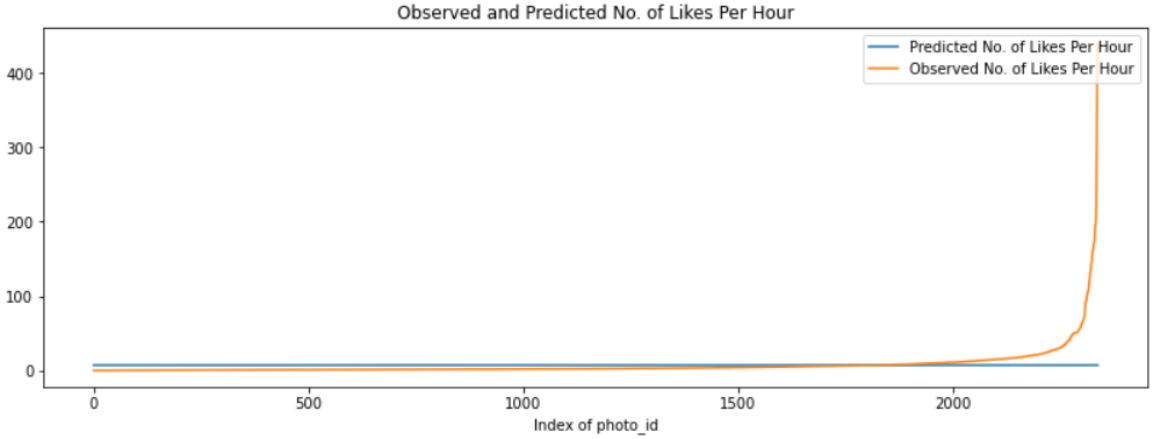
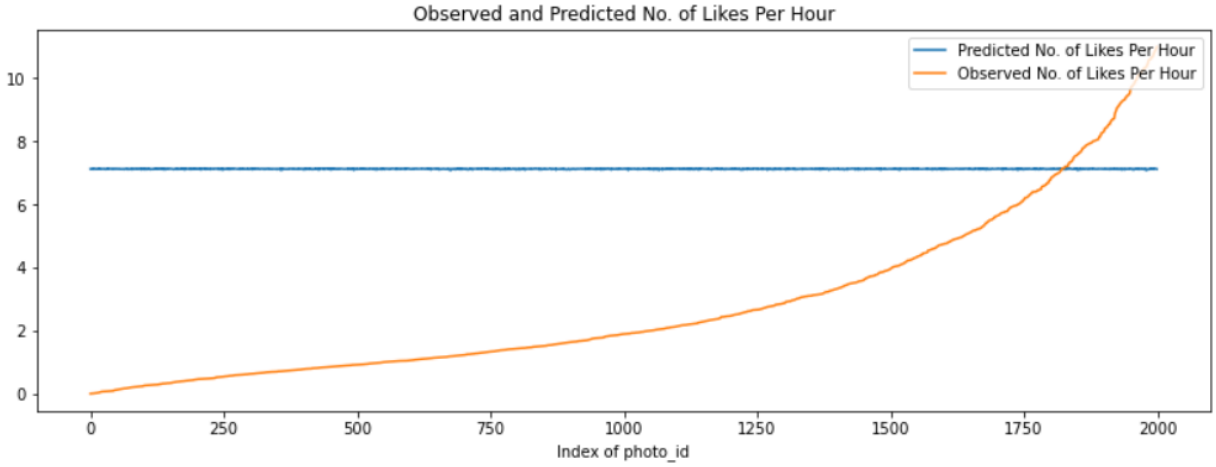


Figure 6. Training and Validation MSE and MAE.

Using the test data that was set aside in the beginning, predictions on the number of likes per hour were made and the results were compared with the actual values. Figure 7 shows the comparison between the predicted and actual number of likes per hour. Here, the test data was sorted in ascending number of likes per hour to have a better visualization of the comparison.



(a) entire test data



(b) excluding the highest values

Figure 7. Predictions on test data vs actual observations on unseen test data

The predictions appear to perform well based on their small distance from the actual observations. However, the presence of some extremely high number of likes per hour causes

the range of the vertical axis to be wide and thus, might have contributed to this visual effect. Additionally, the model seem to perform poorly for extremely high values. Hence, as closer look of the first 2000 photos was taken in order to exclude the extremely high values. On average, the model tends to predict around 7 more likes per hour. Also, the predictions appear to have the same value for all photos and so a closer look is taken as shown in Figure 8.

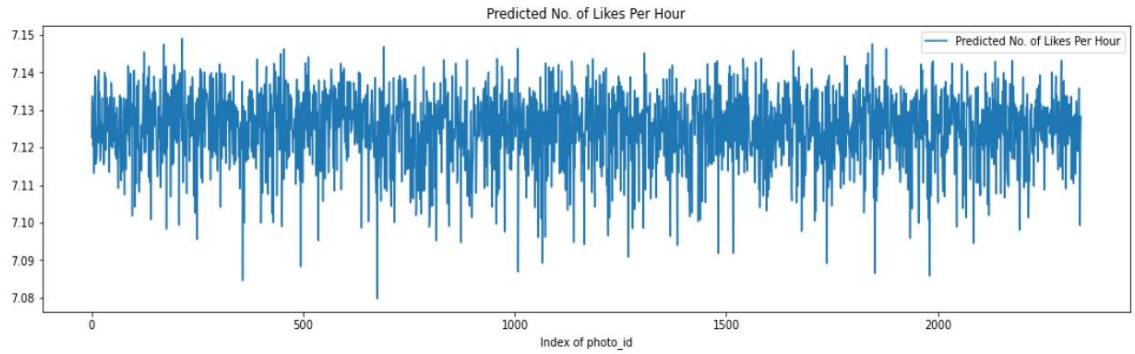


Figure 8. Plot of the model predictions only

It was observed that the predicted number of likes per hour was varying, but the range of variation was small. For this model, there was still much to improve because the results were showing a nearly steady prediction indicating that the model seemed to just give an average of the number of likes per hour and the features of the photo did not seem to enhance the prediction. Hence, fine-tuning was implemented. The MSE and MAE values were plotted as in Figure 9.

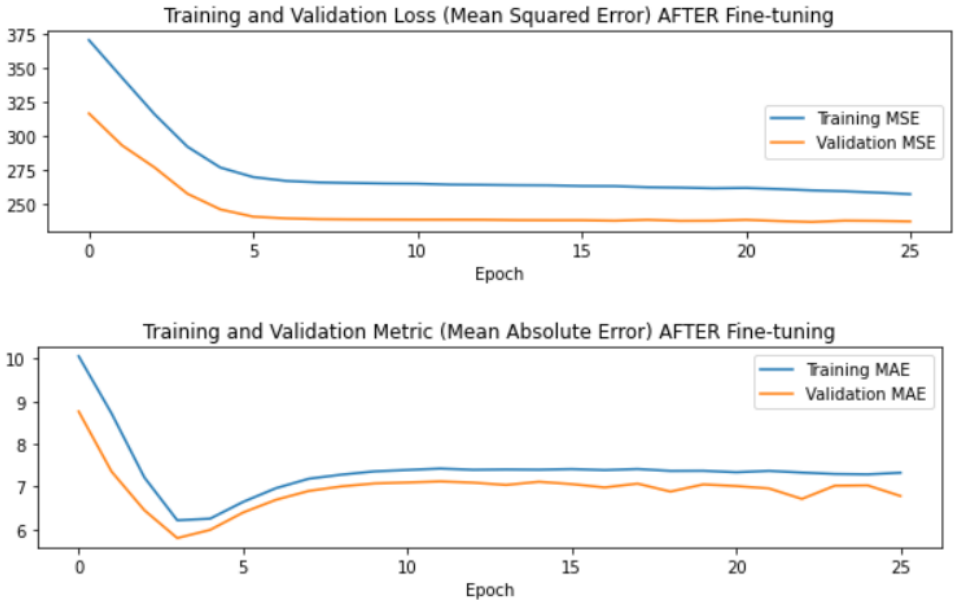
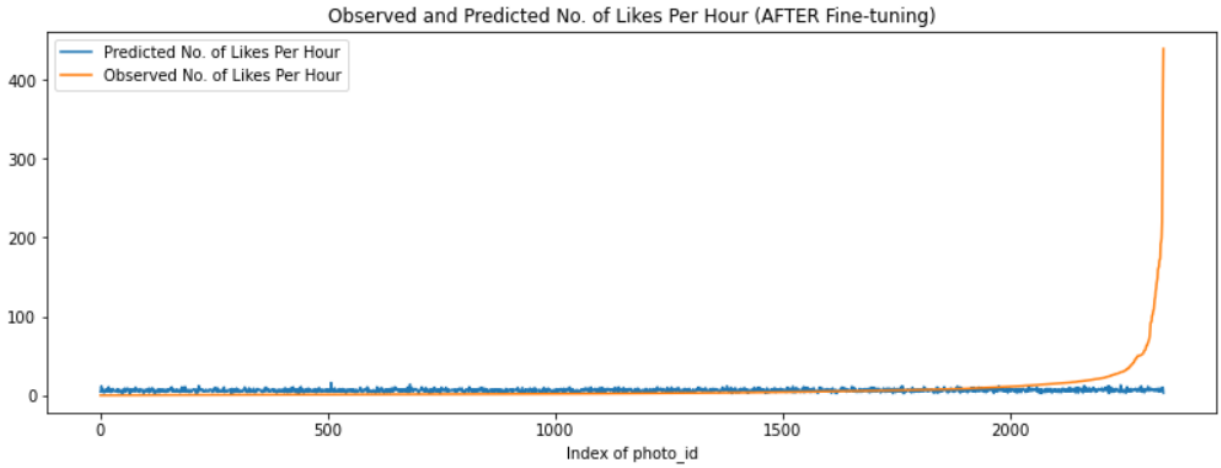
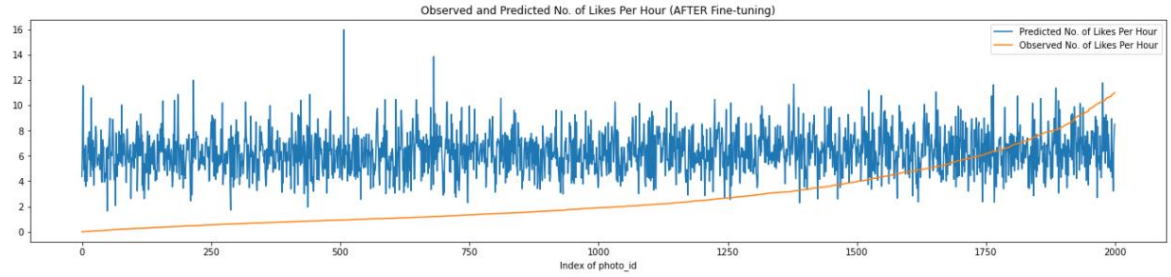


Figure 9. Fine-tuned model MSE and MAE values after 26 epochs

As in the previous model, early stopping was done to avoid overfitting and so after 26 epochs, training stopped because the validation loss was no longer improving even though the training loss was still decreasing. Note that the training error was just slightly larger than the validation error which is a good indicator that overfitting has been taken control of in the current model. Figure 10 plots the predictions of the fine-tuned model in comparison to the actual (unseen) test data.



(a) entire dataset



(b) excluding extreme values

Figure 10. Predictive performance of the fine-tuned model on unseen test data

Here, the variations in predictions are more noticeable compared to the previous model despite the presence of extremely high values. This indicates that the model does not just give

the average number of likes per hour, but it already takes into account the features of the photo to some extent, albeit the performance can still be further improved.

Model's prediction

A randomly selected photo from Instagram was used to test the fine-tuned model (Figure 11). This photo received 5 likes after about an hour (52 minutes to be exact).

Figure 11. Randomly selected Instagram photo to test model prediction performance

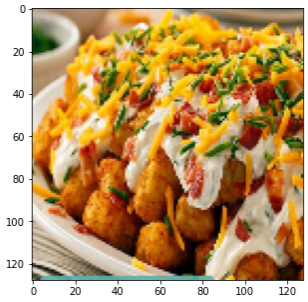
After preparing the photo by resizing it and normalizing its RGB values to form an array (Figure 12), the number of likes in an hour was predicted to be around 5.84 using the fine-tuned model.

Predict number of likes of a new image

In [33]:
```
y_test_new = 5 #actual likes after 1 hour

try:
    img = cv2.imread('C:/Users/Liz/Documents/adv_an/IMG_20210528_001723.jpg')[...,::-1] #read image
    dim = (img_size, img_size)
    resized_img = cv2.resize(img, dim)/255 # Reshaping images to preferred size
except Exception as e:
    print(e)

plt.figure(figsize = (5,5))
plt.imshow(resized_img)
plt.show()
```



In [34]:
```
x_test_new = resized_img.reshape(-1, img_size, img_size, 3)
y_pred_ft_new = model.predict(x_test_new)
y_pred_ft_new
```

Out[34]:
```
array([[5.8446007]], dtype=float32)
```

Figure 12. Predicting the number of likes of a randomly selected Instagram photo

ASSIGNMENT 3: PREDICTING ON STREAMED TEXTUAL DATA

In this assignment, the goal was to build a predictive model via implementing the data processing framework Apache Spark for streaming textual data. In this case, a link that streams tweets in English with specific hashtags—"#vaccine", "#stopasianhate", "#covid", "#china", "#inflation", "#biden"—in real time was provided; the first goal therefore was to collect enough data for a predictive model of the hashtag of the tweet based on its respective text. Second, a script had to be implemented to convert the data into a manipulable dataset—RDD in our case, using pyspark. Finally, after splitting the data, a pipeline for converting tweets and hashtags into a format for a proper predictive model—i.e., multinomial logistic regression—was to be implemented, as well as showcasing how to tune a model's parameters for improved performance.

Spark

Spark is a data processing framework with binding for R, Python, Java, and Scala. It incorporates the use of SQL techniques, streaming, machine learning and others. Spark is often used when dealing with large datasets due to its relative speed and processing efficiency. In this case, the streaming techniques are used as well as the multinomial logistic regression.

The process begins by installing the spark environment done so via the spark.zip file provided on Toledo; Python 3, Anaconda, and Java 11 must be available. Then, it is necessary to ensure that the sample notebooks can run as expected. The process of setting up the environment was not exempt of technical difficulties, considering that Spark requires its own environment and differences between operating systems.

```
py4j.protocol.Py4JJavaError: An error occurred while calling o55.saveAsTextFile.
: org.apache.spark.SparkException: Job aborted.
```

```
Caused by: java.io.IOException: Mkdirs failed to create file:/C:/Users/katherinesarapata/Desktop/spark/notebooks-1622039580000/_temporary/0/_temporary/attempt_20210526163300222460770644937826l_0118_m_000000_0 (exists=false, cwd=file:/Users/katherinesarapata/Desktop/spark/notebooks)
```

After troubleshooting, the provided script was run for streaming tweets. To accomplish the streaming, it is necessary to set up a Spark context which connects and controls Spark clusters

so that the incoming tweets can be converted to resilient distributed datasets, RDDs, which can then be easily processed.

Once the Spark context is set up, the Streaming context is also set up to access continually incoming data from some source, such as a specific URL. For accessing tweets, the Streaming context is told to download batches of tweets in intervals of 10 seconds.

```
ssc = StreamingContext(sc, 10)
```

This streaming context is adapted for streaming texts from a local server, “seppe.net” in this case, which was preconfigured to select tweets with specific hashtags. The batches of streamed tweets are saved in individual folders to be accessed later for analysis. This means a folder is created every ten seconds that may contain zero, one, or more json objects. The tweets are converted to json objects with keys *tweet_id*, *tweet_text*, and *label*. The provided technique for streaming tweets allows for the streaming process to be stopped and started manually.

```
lines = ssc.socketTextStream("seppe.net", 7778)
lines.saveAsTextFiles("file:///C:/Users/ncozz/Desktop/spark/output/myoutput")
```

This stream was run for multiple days, ending up with a total of 206,414 files, 84,985 folders, and 9.19 MB of data, to ensure enough quantity of data for a predictive algorithm.

It is also possible to immediately save incoming data in a data frame using the *readStream*() method rather than creating a streaming context. This is Spark Structured Streaming which has higher efficiency as requires less data processing, as well as allowing for data to be immediately manipulated with scala operations and allowing for smaller batches of streams. Structured Streaming might be more beneficial in a commercial setting; however, for the purposes of this project, generic Spark Streaming is implemented as it is a more straightforward process.

Data Processing

After streaming tweets, a separate notebook is implemented for analyzing the obtained data. A new Spark Context is created and now the data is accessed via an absolute path. The

data is stored as a collection of folders of text files in a json format. The folders of text files are all converted to a single RDD using the method *textFile*(), then the method *read.json*() interprets the schema of the json objects and returns a Spark DataFrame.

```
data = sc.textFile('C:/Users/ncozz/Desktop/spark/output/myoutput-*')
df = spark.read.json(data)
```

Duplicated data and data with N/As are dropped. Below is the distribution of the target hashtags represented by the collected data. A null row is generated for every streamed hashtag which are eventually dropped. The data is then split into a training set of 80% and a testing set of 20% to begin developing a model.

```
+--------------+-----+
|         label|count|
+--------------+-----+
|        #covid| 4329|
|          null|18195|
|        #biden| 2851|
|      #vaccine| 5095|
|        #china| 3976|
|#stopasianhate| 1354|
|    #inflation|  590|
+--------------+-----+
```

A pipeline is created using the *Pipeline command* to efficiently parse the dataset. In the case of the explanatory variables, the written text of each tweet is separated into columns of individual words using *RegexTokenizer*(). Next, the method *StopWordsRemover*() removes commonly used words with little predictive power, such as “and”, “the”, “a”, etcetera. The frequency of each remaining word is found using *CountVectorizer*(), which is saved as *features* and the final input vector.

The frequency of each of the hashtags—the target variable—is found using *StringIndexer*() and saved as *labelindex*.

With the tweet text and hashtags converted, a multinomial logistic regression is defined which uses the *features* to find the *labelindex*—with the initial parameters of a maximum iteration

of 5 in our case. There is also a regularization parameter set 0.3, λ which is the penalty against model complexity that goes into the loss function. There is also an elastic net parameter set at 0, α which controls the weights of the ridge and lasso regressions in the loss function. The logistic regression is defined with the method *LogisticRegression*(). The multinomial logistic regression determines predictions by minimizing the negative log likelihood with penalties to prevent overfitting.

```
# Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and lr.
tokenizer = RegexTokenizer(inputCol="tweet_text", outputCol="tokentweet")
stopWordsRemover = StopWordsRemover().setInputCol("tokentweet").setOutputCol("words")
countVectorizer = CountVectorizer().setInputCol("words").setOutputCol("features")
stringIndexer = StringIndexer(inputCol = "label", outputCol = "labelindex")
lr = LogisticRegression(featuresCol = 'features', labelCol = 'labelindex', maxIter=5, regParam=0.3, elasticNetParam=0.0
indexToString = IndexToString(inputCol = "labelindex", outputCol = "pred_hashtag")

pipeline = Pipeline(stages=[tokenizer, stopWordsRemover, countVectorizer, stringIndexer, lr, indexToString])
```

This pipeline accepts the tweet data, converts it to a usable form, defines a model, and convers the results back into easily interpretable data. The label index of the predicted hashtags get converted from integers to strings after the logistic regression is ran using the *IndexToString*() method.

Model Fitting

The accuracy of the model created by fitting the train data using the pipeline is calculated with *MulticlassClassificationEvaluator*() and weighted precision—fraction of positive classes that are properly predicted as positive—as the accuracy metric. The weighted recall is also examined. For the initial parameters, the accuracy and test error are for the weighted precision are a low 65.4% and thus a test error of 34.6%. An alternative metric is the recall rate—the fraction between true positives versus true positives plus false negatives—which can also be implemented, which gives 62.5%.

The model is saved locally and a parameter grid is created with which to perform 3 fold cross validation to find the parameters with the highest weighted precision accuracy. To do this, the methods *ParamGridBuilder*() and *CrossValidator*() are used.

```
evalcxv = MulticlassClassificationEvaluator(
    labelCol="labelindex", predictionCol="prediction", metricName="weightedPrecision")

paramGrid = ParamGridBuilder() \
    .addGrid(lr.elasticNetParam, [0.3,0.5,0.8]) \
    .addGrid(lr.regParam, [0.1, 0.01, 0.3, 0.5]) \
    .build()

crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=evalcxv,
                          numFolds=3)  # use 3+ folds in practice

# Run cross-validation, and choose the best set of parameters.
cvModel = crossval.fit(train)
```

It is worth mentioning that the cross-validation step can take many hours depending on the dataset size and hardware capacities—in the case of the implemented script, it took about four days straight for the command of *CrossValidator()* to finish executing.

This process is repeated twice—the second time with a much more reduced net—and the found optimal parameters are *elasticNetParam* = 0.3 and *regParam* = 0.1. With this set of parameters, the model has a weighted-precision-accuracy of 0.7151; however, the weighted-recall-precision is 0.4768.

These results illustrate the tradeoff between precision and recall—thus why ROC curves are frequently implemented. In this case, since the model was optimized based on weighted precision, this metric shows improvement while the recall decreases. There are scenarios where the weighted recall would be more important and thus, more relevant for parameter tuning; this should be assessed depending on the goal behind the prediction model.

As the model has been locally saved and has optimized parameters, the model can be loaded into a Jupyter notebook globally and the few lines of processing to be transferred to deal with new incoming tweets.

There are several reasons that our model is not perfectly accurate. Regarding customers, Twitter is used by ordinary people who are likely to make typos which as of now cannot be edited or misuse a hashtag. The language used by the layperson between #china and #stopasianhate has a high potential to overlap, which could complicate predictions. Additionally, the authors of the tweets could be using the hashtags sarcastically, which might difficult for humans to understand and incredibly difficult for a machine analyzing language.

The model could also be improved by collecting the twitter handles of the authors as source of a tweet can impact the meaning. For example, “This country is struggling” would have different predictions coming from an individual from China, a right wing extremist from the US, and a leftwing extremist from the US. Humans can easily predict the respective hashtags #china,

#biden, and #covid, however a machine would struggle to tell the difference based on language alone.

Last but not least, Spark Streaming turns out to have a high computational cost for a regular machine; thus, each command can take several hours to execute, which implies that the actual process of finding an appropriate model could take several days or weeks to have results.

ASSIGNMENT 4: EXPLORING GRAPHS WITH NEO4J AND GEPHI

The general approach behind this assignment is to gain some interesting insights about the increase of disinformation and fake news on YouTube. The insights found in this assignment might be an explanatory factor in this process; this means we propose a certain “theory” based on what one can see from the data patterns and neglect the empirical work to prove the underlying.

Recommendations on known misleading videos

In a first step, we tried to find out what recommended results we get from videos that we know are misleading. To become a more interesting set of results we only used the recommended results that are top results (i.e. position 1):

```
match (r:RECOMMENDED_RESULT)<-[:RECOMMENDED_RESULT]- (v:VIDEO)
WHERE v.is_known_misleading=true
and r.position =1
return r,v
```

Next, we took a look at these recommended results and tried to determine hashtags that are commonly used within these videos; the videos that are recommended based on the statement above. The Cypher query becomes:

```
Match (h:HASHTAG) <-[:MENTIONS]-(v1:VIDEO)<-[:RECOMMENDED_RESULT]-
(r:RECOMMENDED_RESULT)<-[:RECOMMENDED_RESULT]-(v2:VIDEO)
where v2.is_known_misleading = true
and r.position =1
return h
```

Based on the results we got from the statement, we concluded some very interesting things. Some hashtags were often used in videos that were recommended by known misleading videos, especially the hashtags facemask, microscope and microbehunter and worms. Based on these insights we came up with the following proposition: If hashtags are often used in videos that are fake or in videos that are recommended based on fake videos it could be interesting to take a closer look at all videos containing this hashtag or the most significant parts of these

hashtags because the hashtag might flash a fake/misleading video. Besides this proposition it seems to be, when taking a closer look at the titles, that a lot of videos are about what a facemask is made of.

After adding our proposition, we continued the process and found that queries leading to videos with hashtag containing the words “micro”, “inside” or “worm” are almost always biased. Therefore, it might be also interesting to take into account the fact whether a query is biased or not.

```
Match (q:QUERY) --(v:VIDEO)-[:MENTIONS]->(h:HASHTAG)
with h.id AS vocabulary, q As q, v As v
where vocabulary =~'.*micro.*'
or vocabulary =~".*worm.*"
or vocabulary =~".*inside.*"
return vocabulary, q.is_misleading
```

Earlier we showed that hashtags containing the words “micro”, “worm” or “inside” are suspicious and indicators of misleading videos. Also, we showed that they go hand in hand with queries that are misleading and are part of videos that are recommended by videos that we know are misleading.

Therefore, to end this part of our research one may conclude it might be interesting to double check videos resulting from search results that come from a possibly biased and misleading query if this query also leads to videos containing the words “micro”, “worm” or “inside” in their hashtag(s).

```
Match (v2:VIDEO)<-[:SEARCH_RESULT]-(s:SEARCH_RESULT)-[:SEARCH_RESULT]-
(q:QUERY) --(v:VIDEO)-[:MENTIONS]->(h:HASHTAG)
with h.id AS vocabulary, q As q, v As v, s As s,v2 as v2
where vocabulary =~'.*micro.*'
or vocabulary =~".*worm.*"
or vocabulary =~".*inside.*"
and q.is_misleading= true
Return v2
```

Checking this information stream indeed seems to uncover a collection of mysterious, suspicious sounding video titles; “the truth about covid”, “worms on FFP2” , …

Connections to potentially misleading videos

The second part of this assignment further investigates potentially misleading videos that were not classified as such yet. Another focus was to learn more about how misleading videos are embedded in the whole network. Therefore, a new query is created and exported to Gephi. Here, we will check what videos are recommending misleading videos on COVID-19. In order to capture indirect recommendations through paths of videos, we decided to include all videos (and the corresponding channels that uploaded the videos) that indirectly recommend misleading videos through a path of up to 7 videos.

```
CALL apoc.export.graphml.query("
match (v1:VIDEO)-[r1:RECOMMENDS]->(v2:VIDEO)-[r2:RECOMMENDS]->(v3:VIDEO)-
[r3:RECOMMENDS]->(v4:VIDEO)-[r4:RECOMMENDS]->(v5:VIDEO)-[r5:RECOMMENDS]-
>(v6:VIDEO)-[r6:RECOMMENDS]->(v7:VIDEO)
(c1:CHANNEL)-[u1]-(v1), (c2:CHANNEL)-[u2]-(v2),
(c3:CHANNEL)-[u3]-(v3), (c4:CHANNEL)-[u4]-(v4),(c5:CHANNEL)-[u5]-(v5), (c6:CHANNEL)-
[u6]-(v6), (c7:CHANNEL)-[u7]-(v7)
where v8.is_known_misleading = true
    return *
", "subset.graphml",
 {useTypes: true, storeNodeIds:true, readLabels:true})
```

To structure the graph, the Yifan Hu algorithm is used to separate nodes based on their interaction with one another. Strong interaction of a cluster of nodes will make them align closer to each other. Hence, this algorithm is useful to visualize node clusters that strongly interact with each other. In this context, a connection between two videos indicates that a video is recommended by the other one or that the two videos recommend each other. This depends on the direction of the edge, which is visualized through an arrow. The result is shown in Figure 13. Known misinformative videos are colored in green, whilst not misinformative videos are colored in red. Corresponding channels are visualized as black nodes. The figure shows that

misinformative videos are all located fairly close to each other, which indicates that they are commonly recommending each other. However, the cluster of green dots is not as dense as some other clusters of red points that are found in the graph.

The first goal of the graph analysis with Gephi was to investigate other potentially misinformative videos that are not known as misinformative yet. Therefore, videos that are located nearby the misinformative videos in the graph are checked. Here, it becomes apparent that a fair share of the videos nearby misinformative videos are not misinformative themselves, but just share similar words in the video titles. For instance, trustworthy videos about face masks are found to be recommended by misinformative videos. Also, fact checking videos about worms or parasites in face masks are recommended by misleading videos. This indicates already that consuming conspiracist videos YouTube does not mean that a consumer is trapped in a cycle of conspiracist videos, but instead has the chance to gather information from a variety of sources via recommendations. Still, some previously unknown misinformative videos are found nearby green nodes. Those have titles such as ‘Mongellons / Parasieten op mondkapjes and PCR test’. They are edited within the data lab of Gephi to be part of known misinformative videos.

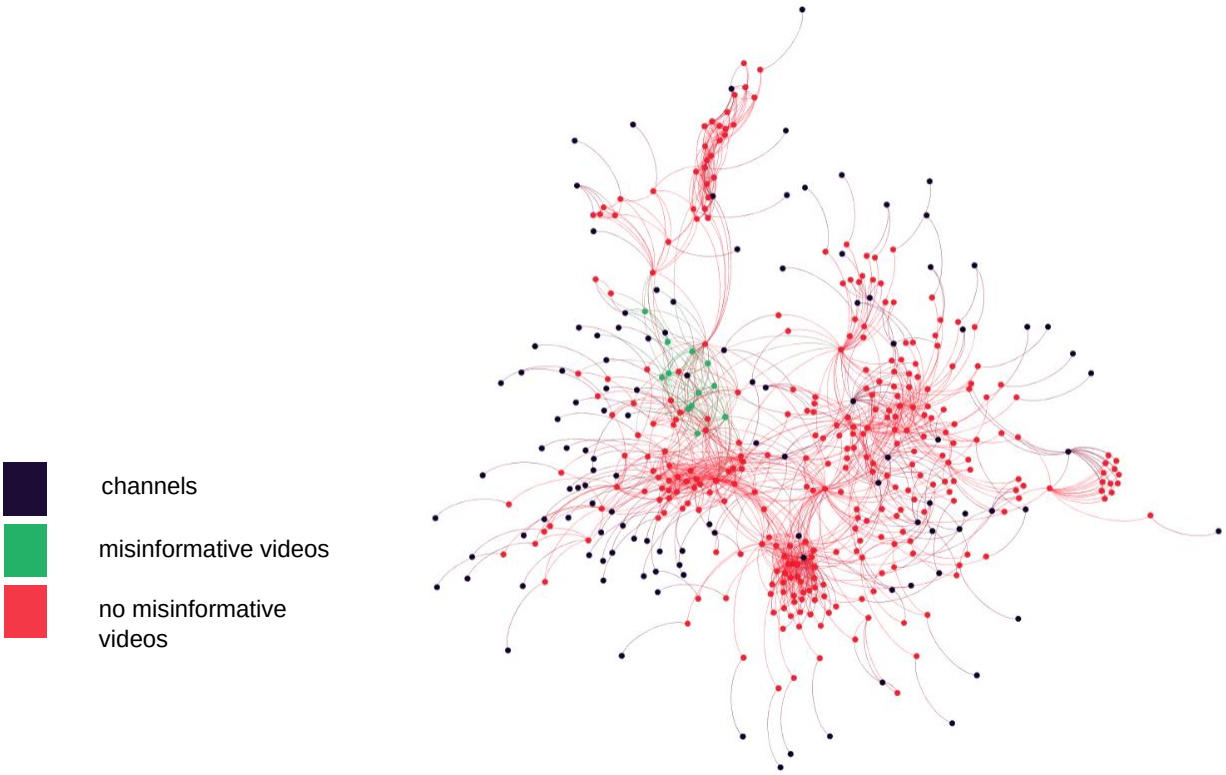


Figure 13. Network of videos and channels directly or indirectly related to misleading COVID-19 videos

Other interesting things to investigate in this graph are if any well-known or popular channels are part of the graph and thus are at least indirectly connected to conspiracy videos. That is a relevant question because YouTube users might be starting to watch informative videos from e.g. news channels on YouTube and then end up with conspiracist videos by just following recommendations. Therefore, the graph is shown with channel labels in Figure 14. The size of the label depends on the number of subscribers the channel has on YouTube. It becomes apparent that a number of well-known large news outlets are included in the graph. Those include BBC News, ABC News, NDTV, NDTV India (cut off on the right side of the graph), and Fox News. In order to investigate how many edges viewers of these channels would have to follow to end up watching a misinformative video, shortest paths between a selection of the news outlets above and misinformative videos are investigated. The results are shown in Table 5. The videos from reputable news outlets that are part of the graph are mostly videos about the spread of Covid and Covid vaccines. When investigating the course of the paths from the news outlets to misinformative videos, two striking observations are made. Firstly, all paths end in a specific misinformative video about parasite in face masks (url: https://www.youtube.com/watch?v=7e9Lt6IkuSA). Secondly, the paths always go over the same final node before arriving at the misinformative video. This node is a video uploaded by the *Montreal Gazette,* a well-respected English-language newspaper from Montreal. The video is about a scientific explanation of graphene face masks (url: https://www.youtube.com/watch?v=2fNKgCO1p4E). This appears to be a key node in linking reputable channels with unreliable ones.

Table 5. Shortest Path from video of selected reputable channels to misinformative video

Videos of Channel	**Shortest Path to misinformative video**
Fox News	*6*
BBC	*6*
ABC News	*6*
BBC News	*6*
Montreal Gazette	*1*

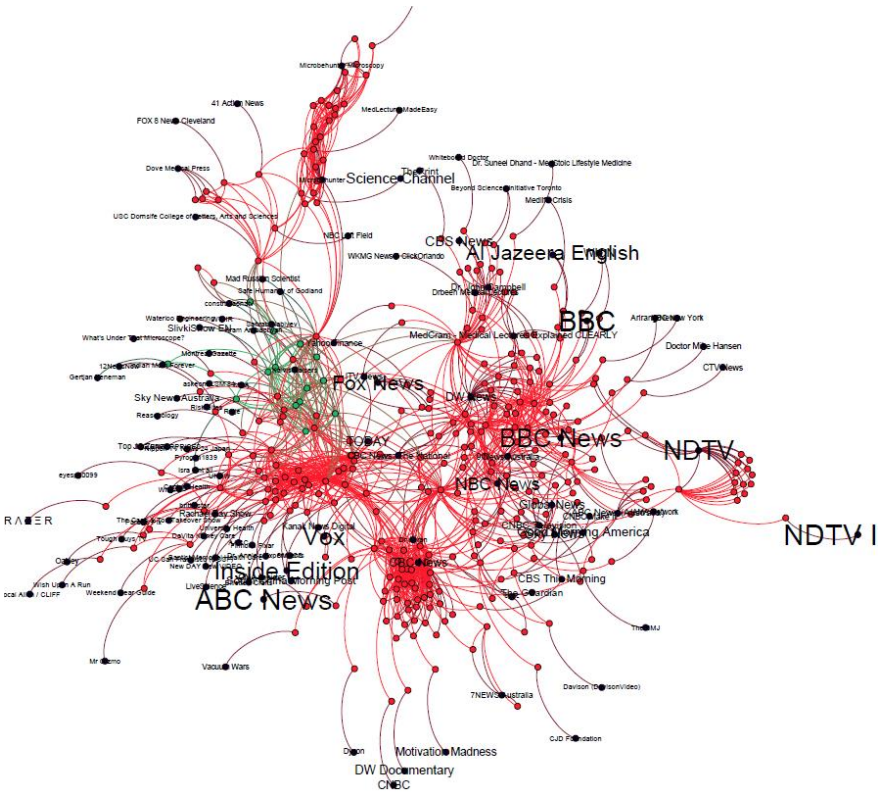


Figure 14. Network of videos and channels directly or indirectly related to misleading COVID-19 videos (with channel labels in relation to number of subscriptions)

Channels that lead to misleading videos

As a last part of the analysis, it will be investigated which channels have the most videos present in the graph. Since only videos that are recommending misinformative videos over up to 7 edges are included, this would be relevant in order to find channels whose contents are prone to be leading to misinformative videos. Therefore, the graph is plotted in Figure 15. A higher number of outgoing edges is shown through a larger label. The figure shows that especially videos from the American-based CBC News and German-based DW News are indirectly connected to misinformative videos. Both of these news outlets are reputable. Other slightly outstanding channels are Microbehunter Microscopy and Microbehunter that upload videos about the use of microscopes. Both are not posting misinformative videos. In fact, Microbehunter published a fact-check video that refutes the claims made in the misinformative videos.

In conclusion, the graph analysis showed that some well-known news outlets are connected to misinformative videos through fairly long paths. As the only reputable outlet, a video of the Montreal Gazette was found to directly recommend a misinformative video.

Figure 15. Network of videos and channels directly or indirectly related to misleading COVID-19 videos (with channel labels in relation to number of outgoing edges)