1 Assignment 1

The goal of this assignment is to predict customer churn of a bank/insurance company, by using the status, product ownership and account balance informations of the customers, based on three datasets from three successive months.

Before finding the best performing model with PyCaret library, we tried one of the approaches in the assignment description; which is labelling customers whose liquidity is reduced by twenty percent or more as churners. This resulted in very poor performance. The second approach was using decision tree regressor and classifier on the liquidity values. Again, it gave a very poor score with a negative R squared value.

[1]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

[2]:
```
from pycaret.regression import setup
import lightgbm
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
```

After some investigation, we found an article which was explaining customer churn prediction using PyCaret. It is a machine learning library that automates machine learning workflows. [1] After preprocessing and cleaning the data, it automatically applies all the algorithms available in the model library and evaluates multiple performance metrics using cross-validation. [2]

We first install the PyCaret library.

[3]:
```
#pip install pycaret
```

We read the training data using pandas library. As the names indicate, "tminus2train" dataset includes the product ownership and balance data from two months ago, "tminus1train" includes the same information from a month ago and finally "ttrainwithtarget" includes the same information at time t with the target values indicating customer churn. We will use those three datasets to train our model on.

[4]:
```
ttrainwithtarget = pd.read_csv('train_month_3_with_target.csv', na_values=['?'])
tminus1train = pd.read_csv('train_month_2.csv', na_values=['?'])
tminus2train = pd.read_csv('train_month_1.csv', na_values=['?'])
```

Then we read the test datasets, to test our model and upload the results on the leaderboard. With the same idea behind, "tminus2test" dataset contains the data from two months ago, "tminus1test" from a month ago and "ttest" contains the state at time t.

[5]:
```
ttest = pd.read_csv('test_month_3.csv', na_values=['?'])
tminus1test = pd.read_csv('test_month_2.csv', na_values=['?'])
tminus2test = pd.read_csv('test_month_1.csv', na_values=['?'])
```

As explained in the assignment description, we created new columns for each dataset, by summing up the balances for each customer. We added those liquidity columns to the dataset with the target values (at time t)("ttrainwithtarget").

[6]:
```
ttrainwithtarget['liquiditytimet'] = ttrainwithtarget['bal_current_account'] + \
    ttrainwithtarget['bal_savings_account']
ttrainwithtarget['liquiditytimetminus1'] = tminus1train['bal_current_account'] +␣
 ↪\
    tminus1train['bal_savings_account']
ttrainwithtarget['liquiditytimetminus2'] = tminus2train['bal_current_account'] +␣
 ↪\
    tminus2train['bal_savings_account']
```

Then we calculate the percentage change by comparing the liquities with time:

- "t-1" with "time t",
- "t-2" with "time t" and
- "t-2" with "t-1".

[7]:
```
ttrainwithtarget['liquidity_change_01'] =␣
 ↪(ttrainwithtarget['liquiditytimetminus1']-ttrainwithtarget['liquiditytimet'])/
 ↪ttrainwithtarget['liquiditytimet']
ttrainwithtarget['liquidity_change_02'] =␣
 ↪(ttrainwithtarget['liquiditytimetminus2']-ttrainwithtarget['liquiditytimet'])/
 ↪ttrainwithtarget['liquiditytimet']
ttrainwithtarget['liquidity_change_12'] =␣
 ↪(ttrainwithtarget['liquiditytimetminus2']- \
                                          ␣
 ↪ttrainwithtarget['liquiditytimetminus1'])/
 ↪ttrainwithtarget['liquiditytimetminus1']
```

We apply the same summations and percentage calculations also to the test datasets.

[8]:
```
ttest['liquiditytimet'] = ttest['bal_current_account'] + \
    ttest['bal_savings_account']
ttest['liquiditytimetminus1'] = tminus1test['bal_current_account'] + \
    tminus1test['bal_savings_account']
ttest['liquiditytimetminus2'] = tminus2test['bal_current_account'] + \
    tminus2test['bal_savings_account']
ttest['liquidity_change_02'] =␣
 ↪(ttest['liquiditytimetminus2']-ttest['liquiditytimet'])/ttest['liquiditytimet']
ttest['liquidity_change_01'] =␣
 ↪(ttest['liquiditytimetminus1']-ttest['liquiditytimet'])/ttest['liquiditytimet']
ttest['liquidity_change_12'] =␣
 ↪(ttest['liquiditytimetminus2']-ttest['liquiditytimetminus1']) / \
ttest['liquiditytimetminus1']
```

As a data preprocessing step, we fill all missing values with the median.

[9]:
```
ttrainwithtarget = ttrainwithtarget.fillna(ttrainwithtarget.median())
tminus1train = tminus1train.fillna(tminus1train.median())
tminus2train = tminus2train.fillna(tminus2train.median())
```

```
#for the test data
ttest = ttest.fillna(ttest.median())
tminus1test = tminus1test.fillna(tminus1test.median())
tminus2test = tminus2test.fillna(tminus2test.median())
```

Considering the importance of children ownership and being in a relationship, those columns are stored as variables, to then create different datasets containing those categorical values ("data_cat" and "data_cattest").

[10]:
```
children = ttrainwithtarget['customer_children']
childrentest = ttest['customer_children']
relationship = ttrainwithtarget['customer_relationship']
relationshiptest = ttest['customer_relationship']
data_cat = pd.concat([children, relationship], axis=1)
data_cattest = pd.concat([childrentest, relationshiptest],axis=1)
```

[11]:
```
data_cat.head()
```

[11]:

	customer_children	customer_relationship
0	NaN	NaN
1	mature	couple
2	NaN	single
3	NaN	NaN
4	mature	couple

Then we edited the columns containing the time information. We only included the year information and excluded the month, by taking the first four characters in the cells. We apply these changes to the train dataset with the target value as well as the test dataset, separately.

[12]:
```
sinceall = ttrainwithtarget['customer_since_all']
sincealltest = ttest['customer_since_all']
sincealldf = pd.DataFrame(sinceall)
sincealldftest = pd.DataFrame(sincealltest)
```

[13]:
```
sincealldf.head()
```

[13]:

	customer_since_all
0	1983-03
1	2017-01
2	1980-12
3	1998-08
4	2012-11

[14]:
```
ttrainwithtarget['customer_since_allyear'] =␣
↪ttrainwithtarget['customer_since_all'].str[:4]
```

[15]:
```
ttrainwithtarget['customer_since_bankyear'] =␣
↪ttrainwithtarget['customer_since_bank'].str[:4]
```

[16]:
```
ttrainwithtarget['customer_birth_dateyear'] =␣
↪ttrainwithtarget['customer_birth_date'].str[:4]
```

[17]:
```
ttest['customer_since_allyear'] = ttest['customer_since_all'].str[:4]
ttest['customer_since_bankyear'] = ttest['customer_since_bank'].str[:4]
ttest['customer_birth_dateyear'] = ttest['customer_birth_date'].str[:4]
```

[18]:
```
pd.set_option('display.max_columns', 10)
ttest.head()
```

[18]:

	client_id	homebanking_active	has_homebanking	\
0	ccf4cd93d5c32cd8a59809d54b4d53ac	0	0	
1	56605a660d18549592653ff6941186f1	0	0	
2	bda5f84c05e5695a7ec10550b457890f	0	0	
3	a2f1c04bc3acf2222e658a897400798f	0	0	
4	e83aadc3b0d25dbc12a35551afa25807	0	0	

	has_insurance_21	has_insurance_23	...	liquidity_change_01	\
0	0	0	...	-0.048673	
1	0	0	...	0.001999	
2	0	0	...	-0.115190	
3	0	0	...	-0.013986	
4	0	0	...	0.000000	

	liquidity_change_12	customer_since_allyear	customer_since_bankyear	\
0	0.124031	1981	1981	
1	0.000000	1993	1993	
2	0.001431	1993	2005	
3	-0.015474	1978	2009	
4	0.000000	2014	2014	

	customer_birth_dateyear
0	1937
1	1941
2	1969
3	1952
4	1962

[5 rows x 48 columns]

We turn categorical variables into numerical dummies and then fill missing values with the median.

[19]:
```
data_cat = pd.get_dummies(data_cat)
data_cattest = pd.get_dummies(data_cattest)
```

[20]:
```
data_cat = data_cat.fillna(data_cat.median())
data_cattest = data_cattest.fillna(data_cattest.median())
```

[21]:
```
data_cat.head()
```

[21]:

	customer_children_adolescent	customer_children_grownup	\
0	0	0	
1	0	0	
2	0	0	
3	0	0	
4	0	0	

	customer_children_mature	customer_children_no	customer_children_onebaby	\
0	0	0	0	
1	1	0	0	
2	0	0	0	
3	0	0	0	
4	1	0	0	

	customer_children_preschool	customer_children_yes	\
0	0	0	
1	0	0	
2	0	0	
3	0	0	
4	0	0	

	customer_children_young	customer_relationship_couple	\
0	0	0	
1	0	1	
2	0	0	
3	0	0	
4	0	1	

	customer_relationship_single
0	0
1	0
2	1
3	0
4	0

We turned "year" values into "float64" type for both of the datasets.

[22]:
```
ttrainwithtarget['customer_birth_dateyear'] =␣
 ↪ttrainwithtarget['customer_birth_dateyear'].astype(
    'float64')
ttrainwithtarget['customer_since_allyear'] =␣
 ↪ttrainwithtarget['customer_since_allyear'].astype(
    'float64')
ttrainwithtarget['customer_since_bankyear'] =␣
 ↪ttrainwithtarget['customer_since_bankyear'].astype(
```

```
    'float64')
```

[23]:
```
ttest['customer_birth_dateyear'] = ttest['customer_birth_dateyear'].
 ↪astype('float64')
ttest['customer_since_allyear'] =  ttest['customer_since_allyear'].
 ↪astype('float64')
ttest['customer_since_bankyear'] = ttest['customer_since_bankyear'].
 ↪astype('float64')
```

[24]:
```
ttrainwithtarget.head()
```

[24]:

	client_id	homebanking_active	has_homebanking	\
0	910df42ad36243aa4ce16324cd7b15b0	0	0	
1	4e19dc3a54323c5bbfc374664b950cd1	1	1	
2	f5d08db1b86c0cb0f566bf446cff1fb4	1	1	
3	26170ecf63653e215c52f4262c1c4859	0	0	
4	c078009957dffb64f20e61b41220a976	0	0	

	has_insurance_21	has_insurance_23	...	liquidity_change_02	\
0	0	0	...	-0.072156	
1	0	0	...	-0.072980	
2	0	0	...	0.062157	
3	0	0	...	0.000000	
4	0	0	...	-0.014652	

	liquidity_change_12	customer_since_allyear	customer_since_bankyear	\
0	-0.052013	1983.0	1994.0	
1	-0.059912	2017.0	2017.0	
2	0.012195	1980.0	1980.0	
3	0.000000	1998.0	2013.0	
4	-0.007380	2012.0	2012.0	

	customer_birth_dateyear
0	1943.0
1	1994.0
2	1936.0
3	1946.0
4	1996.0

[5 rows x 49 columns]

Now the columns we are interested in, are all in numerical format, so we can exclude object types and drop "target" column.

[25]:
```
num = ttrainwithtarget.select_dtypes(exclude=["object"]).columns.drop("target")
data_num = ttrainwithtarget[num]
```

[26]:
```
numtest =       ttest.select_dtypes(exclude=["object"]).columns
data_numtest =  ttest[num]
```

[27]:
```
data_numtest.head()
```

[27]:
```
   homebanking_active  has_homebanking  has_insurance_21  has_insurance_23  \
0                   0                0                 0                 0
1                   0                0                 0                 0
2                   0                0                 0                 0
3                   0                0                 0                 0
4                   0                0                 0                 0

   has_life_insurance_fixed_cap  ...  liquidity_change_02  \
0                             0  ...             0.069322
1                             0  ...             0.001999
2                             0  ...            -0.113924
3                             0  ...            -0.029243
4                             0  ...             0.000000

   liquidity_change_12  customer_since_allyear  customer_since_bankyear  \
0             0.124031                  1981.0                   1981.0
1             0.000000                  1993.0                   1993.0
2             0.001431                  1993.0                   2005.0
3            -0.015474                  1978.0                   2009.0
4             0.000000                  2014.0                   2014.0

   customer_birth_dateyear
0                   1937.0
1                   1941.0
2                   1969.0
3                   1952.0
4                   1962.0

[5 rows x 42 columns]
```

We again, fill missing values with the median.

[28]:
```
ttrainwithtarget = ttrainwithtarget.fillna(ttrainwithtarget.median())
ttest = ttest.fillna(ttest.median())
```

Now we can append numerical values with the categorical values, which are already in dummy numerical type. The combined dataset is called "X" and it will be used to train our algorithm.

[29]:
```
X = pd.concat([data_num, data_cat], axis=1)
Xtest = pd.concat([data_numtest, data_cattest], axis=1)
y = ttrainwithtarget['target']
```

[30]:
```
X = pd.DataFrame(X)
y = pd.DataFrame(y)
```

[31]:
```
print(X.head(),"\n\n",y.head())
```

```
   homebanking_active  has_homebanking  has_insurance_21  has_insurance_23  \
0                   0                0                 0                 0
1                   1                1                 0                 0
2                   1                1                 0                 0
3                   0                0                 0                 0
4                   0                0                 0                 0

   has_life_insurance_fixed_cap  ...  customer_children_preschool  \
0                             0  ...                            0
1                             0  ...                            0
2                             0  ...                            0
3                             0  ...                            0
4                             0  ...                            0

   customer_children_yes  customer_children_young  \
0                      0                        0
1                      0                        0
2                      0                        0
3                      0                        0
4                      0                        0

   customer_relationship_couple  customer_relationship_single
0                             0                            0
1                             1                            0
2                             0                            1
3                             0                            0
4                             1                            0

[5 rows x 52 columns]

    target
0       0
1       0
2       0
3       0
4       1
```

We specify the boolean columns as type boolean and then check if the data types are correct.

[32]:
```
X['homebanking_active'] = X['homebanking_active'].astype('boolean')
X['has_homebanking'] = X['has_homebanking'].astype('boolean')
X['has_insurance_21'] = X['homebanking_active'].astype('boolean')
X['has_insurance_23'] = X['homebanking_active'].astype('boolean')
```

```
X['has_life_insurance_fixed_cap'] = X['homebanking_active'].astype('boolean')
X['has_life_insurance_decreasing_cap'] = X['homebanking_active'].astype(
    'boolean')
X['has_fire_car_other_insurance'] = X['homebanking_active'].astype('boolean')
X['has_personal_loan'] = X['homebanking_active'].astype('boolean')
X['has_mortgage_loan'] = X['homebanking_active'].astype('boolean')
X['has_current_account'] = X['homebanking_active'].astype('boolean')
```

[33]:
```
Xtest['homebanking_active'] =                  Xtest['homebanking_active'].
 ↪astype('boolean')
Xtest['has_homebanking'] =                     Xtest['has_homebanking'].
 ↪astype('boolean')
Xtest['has_insurance_21'] =                    Xtest['homebanking_active'].
 ↪astype('boolean')
Xtest['has_insurance_23'] =                    Xtest['homebanking_active'].
 ↪astype('boolean')
Xtest['has_life_insurance_fixed_cap'] =        Xtest['homebanking_active'].
 ↪astype('boolean')
Xtest['has_life_insurance_decreasing_cap'] =   Xtest['homebanking_active'].
 ↪astype('boolean')
Xtest['has_fire_car_other_insurance'] =        Xtest['homebanking_active'].
 ↪astype('boolean')
Xtest['has_personal_loan'] =                   Xtest['homebanking_active'].
 ↪astype('boolean')
Xtest['has_mortgage_loan'] =                   Xtest['homebanking_active'].
 ↪astype('boolean')
Xtest['has_current_account'] =                 Xtest['homebanking_active'].
 ↪astype('boolean')
```

[34]:
```
X.dtypes
```

[34]:
```
homebanking_active                  boolean
has_homebanking                     boolean
has_insurance_21                    boolean
has_insurance_23                    boolean
has_life_insurance_fixed_cap        boolean
has_life_insurance_decreasing_cap   boolean
has_fire_car_other_insurance        boolean
has_personal_loan                   boolean
has_mortgage_loan                   boolean
has_current_account                 boolean
has_pension_saving                    int64
has_savings_account                   int64
has_savings_account_starter           int64
has_current_account_starter           int64
bal_insurance_21                      int64
```

```
bal_insurance_23                      int64
cap_life_insurance_fixed_cap          int64
cap_life_insurance_decreasing_cap     int64
prem_fire_car_other_insurance         int64
bal_personal_loan                     int64
bal_mortgage_loan                     int64
bal_current_account                   int64
bal_pension_saving                    int64
bal_savings_account                   int64
bal_savings_account_starter           int64
bal_current_account_starter           int64
visits_distinct_so                  float64
visits_distinct_so_areas            float64
customer_gender                       int64
customer_postal_code                  int64
customer_occupation_code            float64
customer_self_employed                int64
customer_education                  float64
liquiditytimet                        int64
liquiditytimetminus1                  int64
liquiditytimetminus2                  int64
liquidity_change_01                 float64
liquidity_change_02                 float64
liquidity_change_12                 float64
customer_since_allyear              float64
customer_since_bankyear             float64
customer_birth_dateyear             float64
customer_children_adolescent          uint8
customer_children_grownup             uint8
customer_children_mature              uint8
customer_children_no                  uint8
customer_children_onebaby             uint8
customer_children_preschool           uint8
customer_children_yes                 uint8
customer_children_young               uint8
customer_relationship_couple          uint8
customer_relationship_single          uint8
dtype: object
```

To compare models with PyCaret, we set the initial PyCaret function up by indicating "regression", our training dataset and the target column.

[35]:
```
from pycaret.datasets import get_data
import pycaret.regression
X2 = pd.concat([X,y],axis=1)
s = pycaret.regression.setup(data=X2,target="target")
```

	Description	Value
0	session_id	946
1	Target	target
2	Original Data	(63697, 53)
3	Missing Values	True
4	Numeric Features	36
5	Categorical Features	16
6	Ordinal Features	False
7	High Cardinality Features	False
8	High Cardinality Method	None
9	Transformed Train Set	(44587, 41)
10	Transformed Test Set	(19110, 41)
11	Shuffle Train-Test	True
12	Stratify Train-Test	False

We find the best model by applying "compare_models" function in PyCaret library. This function also evaluates performance using cross-validation. We can see that the model with the lowest performance metrics is Gradient Boosting Regressor with MAE 0.0563, MSE 0.0283, RMSE 0.1681, R2 0.0286 and RMSLE 0.1172.

[36]:
```
best_model = pycaret.regression.compare_models()
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
gbr	Gradient Boosting Regressor	0.0563	0.0283	0.1681	0.0286	0.1172	0.9370	2.6190
br	Bayesian Ridge	0.0573	0.0285	0.1687	0.0224	0.1174	0.9475	0.0590
ridge	Ridge Regression	0.0574	0.0285	0.1687	0.0223	0.1175	0.9466	0.0280
lar	Least Angle Regression	0.0574	0.0285	0.1687	0.0222	0.1175	0.9466	0.0360
lr	Linear Regression	0.0574	0.0285	0.1687	0.0219	0.1175	0.9471	0.4300
omp	Orthogonal Matching Pursuit	0.0574	0.0285	0.1688	0.0210	0.1175	0.9493	0.0250
en	Elastic Net	0.0574	0.0287	0.1692	0.0168	0.1177	0.9531	0.0260
huber	Huber Regressor	0.0573	0.0287	0.1692	0.0168	0.1177	0.9528	0.4370
lasso	Lasso Regression	0.0574	0.0287	0.1692	0.0168	0.1177	0.9532	0.0260
lightgbm	Light Gradient Boosting Machine	0.0564	0.0287	0.1692	0.0155	0.1186	0.9336	0.2580
ada	AdaBoost Regressor	0.0572	0.0287	0.1693	0.0149	0.1181	0.9456	0.2720
llar	Lasso Least Angle Regression	0.0583	0.0292	0.1706	-0.0002	0.1186	0.9699	0.0260
dummy	Dummy Regressor	0.0583	0.0292	0.1706	-0.0002	0.1186	0.9699	0.0220
rf	Random Forest Regressor	0.0627	0.0301	0.1733	-0.0324	0.1239	0.9209	9.8010
et	Extra Trees Regressor	0.0635	0.0305	0.1745	-0.0470	0.1252	0.9186	5.8090
knn	K Neighbors Regressor	0.0564	0.0341	0.1845	-0.1705	0.1346	0.9515	3.2730
par	Passive Aggressive Regressor	0.0956	0.0372	0.1927	-0.2759	0.1318	1.0025	0.0420
dt	Decision Tree Regressor	0.0621	0.0621	0.2492	-1.1386	0.1727	0.9074	0.2010

[37]:
```
best_model
```

[37]:
```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
```

```
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=100,
                                n_iter_no_change=None, presort='deprecated',
                                random_state=946, subsample=1.0, tol=0.0001,
                                validation_fraction=0.1, verbose=0, warm_start=False)
```

To tune hyperparameters, we used "tune_model" function from PyCaret.

[38]:
```
tuned_best_model = pycaret.regression.tune_model(best_model)
```

Split	Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
CV-Val	0	0.0538	0.0256	0.1599	0.0240	0.1117	0.9396
	1	0.0581	0.0301	0.1736	0.0291	0.1207	0.9421
	2	0.0551	0.0270	0.1644	0.0232	0.1148	0.9421
	3	0.0579	0.0303	0.1740	0.0312	0.1210	0.9391
	4	0.0562	0.0283	0.1681	0.0234	0.1173	0.9431
	5	0.0604	0.0322	0.1795	0.0334	0.1246	0.9392
	6	0.0557	0.0268	0.1638	0.0308	0.1144	0.9354
	7	0.0565	0.0289	0.1700	0.0293	0.1184	0.9407
	8	0.0560	0.0287	0.1694	0.0368	0.1177	0.9385
	9	0.0534	0.0251	0.1584	0.0264	0.1107	0.9380
	Mean	0.0563	0.0283	0.1681	0.0288	0.1171	0.9398
	Std	0.0020	0.0021	0.0063	0.0043	0.0041	0.0022
Train	nan	0.0557	0.0276	0.1662	0.0523	0.1154	0.9285

[39]:
```
tuned_best_model
```

[39]:
```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.05, loss='ls', max_depth=3,
                          max_features='log2', max_leaf_nodes=None,
                          min_impurity_decrease=0.2, min_impurity_split=None,
                          min_samples_leaf=2, min_samples_split=4,
                          min_weight_fraction_leaf=0.0, n_estimators=160,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=946, subsample=0.4, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
```

We predicted the target value for the test data by both using the best model and the best model with the tuned hyperparameters. [3]

[40]:
```
ytest = pycaret.regression.predict_model(best_model,data=Xtest)
ytest2 = pycaret.regression.predict_model(tuned_best_model,data=Xtest)
```

```
ttestforid = pd.read_csv('test_month_3.csv', na_values=['?'])
ttestforid['client_id']
ytestdf = pd.concat([ttestforid['client_id'],ytest['Label']],axis=1)
ytest2df = pd.concat([ttestforid['client_id'],ytest2['Label']],axis=1)
```

After pasting the "test id"s next to the predicted target, we are ready to submit for the leaderboard.

[41]:
```
ytestdf.to_csv('f2pycaret_regression_results_withliquiditychange.
 ↪csv',index=False,header=True)
ytest2df.to_csv('f2pycaret_regression_results_withliquiditychange_tuned.
 ↪csv',index=False,header=True)

print(sum(ytest['Label']),sum(ytest2['Label']))
```

```
816.3100482866516 817.7255523647723
```

1.1 Reflections

When we submit our predictions on the leader board, we were first in the 16th place in the public rank, with 36 correct predictions out of the first 250 customer churns, and AUC of 0.75795382.

On the hidden test data, our rank increased to the 9th place, however our score decreased to 34 correct predictions.

The reason for the increased rank, despite the decreased score might be, we were focused more on our AUC score, rather than the number of correct predictions. With the second best AUC score on the leader board, our model performs more consistently than the models focused on the correct predictions.

We used all of the information in the data set and that may confused our model. If we had more time, we could try removing some features and compare the performance. Also it would be better if we had more months to train our model on.

1.2 References

[1] PyCaret (2022) https://www.pycaret.org

[2] Predict Customer Churn the Right Way Using PyCaret (2022) www.towardsdatascience.com/predict-customer-churn-the-right-way-using-pycaret-8ba6541608ac

[3] PyCaret Regression Tutorial (2022) www.github.com/pycaret/pycaret/blob/master/tutorials/Regression%20Tutorial%20Level%20Beginner%20-%20REG101.ipynb

2 Assignment 2

2.1 Introduction

The goal of the second assignment is to detect swimming pools using satellite images. More specifically, we chose to construct a deep learning model that located the swimming pool in images that contained one using a bounding box. The dataset we received contained more than 14.000 images each containing a swimming pool. In addition to the images, we also received a json-file with coordinates for the bounding boxes as well as the coordinates of the location. The entire dataset contained 14964 images. During the first couple of tries of training the model using the entire dataset, we noticed two problems. Firstly, some of the laptops used were not able to handle the workload and kept crashing. Secondly, it took a very long time to train the model using the entire dataset. For these reasons we opted to take a subset of the dataset to train our model. By taking a subset we were able to successfully build a model as well as build the model quicker. For our model we used 500 images. In addition, we also took other memory saving approaches to ensure that our laptops would not crash.

The process for this assignment is as follows. The first step is to extract the data needed for this project. As mentioned before, we received a json file with the coordinates however we only needed the X and Y coordinates and not the longitude and latitude. The second step, we need to do a bit of pre-processing before training the model. The main thing is this step was to resize the image to 512 pixels. The third step is to build the model. Lastly, we will evaluate the model.

2.2 Libraries

For the assignment the following libraries were used.

[1]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from matplotlib import pyplot as plt
import cv2
import json
from matplotlib.patches import Rectangle

from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split

from tensorflow.keras.metrics import MeanSquaredError
from tensorflow.keras.callbacks import EarlyStopping
```

2.3 Data

[2]:
```
PATH = ''
NUMBER_TO_LOAD = 500  # for whole dataset, use 14964
MUTE = True
IMG_SIZE = (128,128)
INPUT_SHAPE = (128,128,3)
```

[3]:
```
with open(PATH + 'metadata.json') as f:
    y_dict = json.load(f)
```

The first thing we do is to check if all the images actually have a pool.

[4]:
```
count = 0
for i in y_dict.keys():
    if y_dict[i]['has_pool'] == False:
        count += 1

if count == 0:
    print("All images have swimming pools!")
```

```
All images have swimming pools!
```

Check if all the names in the json file match with an .png image in our dataset. Then we save the valid .png names into a list for our model.

[5]:
```
png_name = []
for i in list(y_dict.keys())[:NUMBER_TO_LOAD]:
    try:
        img = plt.imread(PATH+'images/'+i)
    except:
        if MUTE:
            pass
        else:
            print(f"cannot find the image {i}")
    else:
        png_name.append(i)
print(f"{len(png_name)} images are valid out of {NUMBER_TO_LOAD}!")
```

```
448 images are valid out of 500!
```

Split image names into train, validation and test set ***1st memory control approach:*** We can save memory by doing the train test split only on the list of image names we just created, and load images when we need to use them.

[6]:
```
X_train, X_test = train_test_split(png_name, test_size=0.2)
X_train, X_val = train_test_split(X_train, test_size=0.25)
```

2.4 Image loading and preprocessing

Here we would like to load and process the image, and then do some visualizations to compare bounding-boxes detected by openCv and original polygon boundaries obtained by using the coordinates from the json file.

Functions for loading and processing images Here we create an one-step function to provide data for visualization as well as for model building. The process is as follows: - Read and resize the image. Resizing into a small trainable size before saving is the ***2nd memory control approach*** - Extract and compute the optimal rectangle from the polygon boundaries. We apply boundingRect from opencv to find the bounding rectangle. - Plot the image. It will return a dataframe with both images and unscaled labels, while for modelling, it will return images with labels scaled into [0,1].

[7]:
```
# TRY cv2.minAreaRect from https://docs.opencv.org/3.1.0/dd/d49/
 ↪tutorial_py_contour_features.html  7.b
def load_resize_img(y_dict, png_list, path=PATH, image_size=IMG_SIZE,␣
 ↪plot=False):
    '''
    to plot: image_size=(512,512), plot=True
    to train: image_size=(IMG_SIZE), plot=False
    '''

    image = []
    label = []

    for i in png_list:
        img = plt.imread(PATH+'images/'+i)

        image.append(cv2.resize(img,image_size))

        coord = np.array([[j['x'],j['y']] for j in y_dict[i]['bounds_x_y']], np.
 ↪float32)
        x,y,w,h = cv2.boundingRect(coord)
        label.append([x, y, x+w, y+h])

    if plot:
        return pd.DataFrame({'png_name':png_list, 'image':image, 'label':label})
    else:
        df = pd.DataFrame({'image':image, 'label':label})
        del image, label
        X = np.stack([row['image'] for index, row in df.iterrows()])
        y = np.stack([list(map(lambda x: x/512, row['label'])) for index, row in␣
 ↪df.iterrows()])
        dflist = [df]
        del df, dflist

        return X, y
```

Comparisons of polygon and rectangular labels Using the the function we just created, we take a first look at the images with the bounding boxes. The blue Polygons are the ones created using the coordinates in the json file. The red bounding boxes are the ones detected by opencv.

[8]:
```
def plot_img(df, n_to_plot=10):
    df_plot = df.loc[:n_to_plot, :]

    fig, axs = plt.subplots(1, n_to_plot, figsize=(80, 80* n_to_plot))

    for i in range(n_to_plot):
        axs[i].imshow(df_plot.iloc[i,1])
        xs, ys = zip(*[(i['x'], i['y']) for i in y_dict[df_plot.
↪iloc[i,0]]['bounds_x_y']])
        axs[i].plot(xs, ys, linewidth=5)
        x1, y1, x2, y2 = df_plot.iloc[i,2]
        axs[i].
↪add_patch(Rectangle((x1,y1),x2-x1,y2-y1,linewidth=5,edgecolor='r',facecolor='none'))
        axs[i].axis("off")

    plt.show()
```

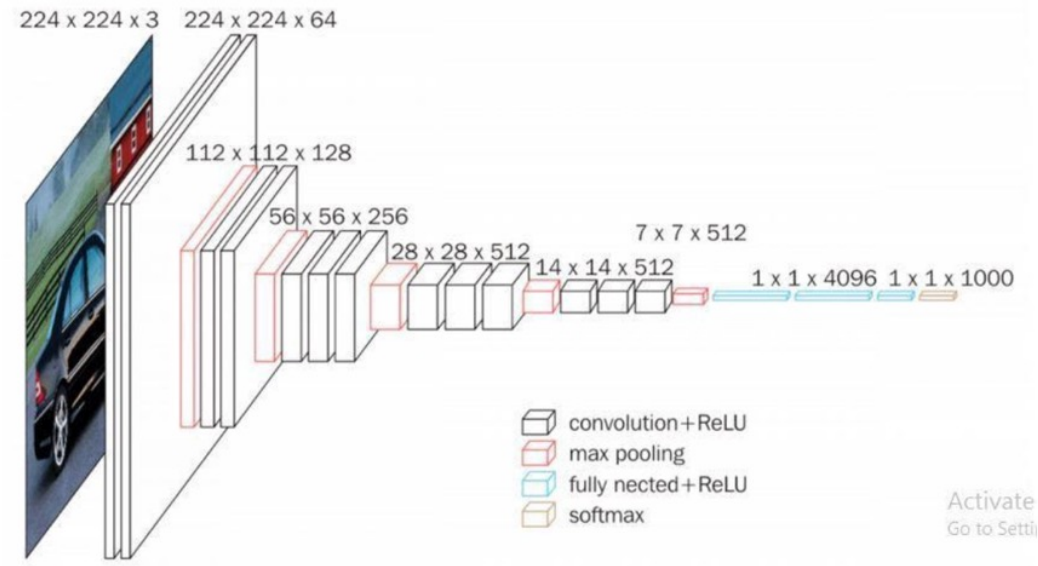
[9]:
```
df = load_resize_img(y_dict, X_train, path=PATH, image_size=(512,512), plot=True)
plot_img(df, n_to_plot = 10)

dflist = [df]
del df, dflist
```

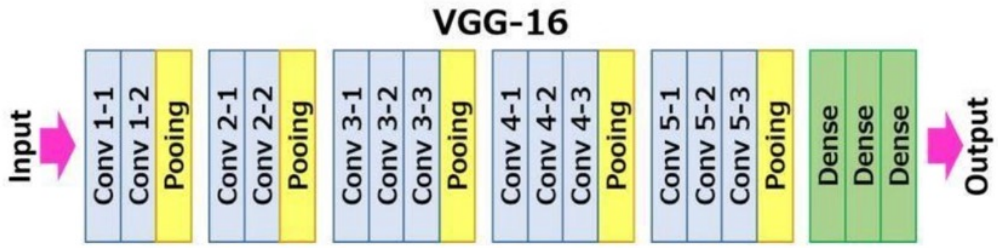
We notice that the opencv is able to detect the swimming pools however the bounding boxes are quite large.

2.5 Model Building

function of images and labels for the model Given the dataset for training is still time and resource demanding, inspired by this link, we tend to use VGG16 with weights pre-trained on ImageNet as our model. VGG16 is a convolution neural network which is considered to be one of the excellent vision model architecture till date. It follows an arrangement of convolution and max pool layers consistently throughout the whole architecture. The image below shows a visual representation of the VGG16 model.

Meaningful initial weights can speed up the training process with less epochs. We also define a method to plot the top 5 best and worst performing images.





class of the model

[10]:
```
class VGG_detection():
    '''
    https://pyimagesearch.com/2020/10/05/
↪object-detection-bounding-box-regression-with-keras-tensorflow-and-deep-learning/
↪
    '''
    def __init__(self, params, verbose=False):
        self.params = params
        self.verbose = verbose


    def get_vgg(self):
        input_shape = self.params['input_shape']    # (img_height, img_width,␣
↪img_channels)
        weights = self.params['weights']
```

```
        include_top = self.params['include_top']
        vgg = VGG16(
            weights=weights,    # None if random initialization
            include_top=include_top,
            input_tensor=Input(shape=input_shape))

        vgg.trainable = False       # freeze all VGG layers so they will *not*␣
↪be updated during the training process
        flatten = vgg.output        # flatten the max-pooling output of VGG
        flatten = Flatten()(flatten)
        x = Dense(128, activation="relu")(flatten)   # construct a␣
↪fully-connected layer header to output the predicted bounding box coordinates
        x = Dense(64, activation="relu")(x)
        x = Dense(32, activation="relu")(x)
        x = Dense(4, activation="sigmoid")(x)

        self.model = Model(inputs=vgg.input, outputs=x)
        return self


    def fit_vgg(self, X_train, X_val, y_train, y_val):
        optimizer = self.params['optimizer']
        loss = self.params['loss']
        metrics = self.params['metrics']
        batch_size = self.params['batch_size']
        epochs = self.params['epochs']
        verbose = self.verbose
        save_name = self.params['save_name']
        monitor = self.params['monitor']

        self.model.compile(loss=loss, optimizer=optimizer,␣
↪metrics=[eval(metrics)])
        earlystop = EarlyStopping(monitor=monitor, patience=5)

        self.history = self.model.fit(X_train, y_train,
                                       validation_data=(X_val, y_val),
                                       batch_size=batch_size,
                                       epochs=epochs,
                                       verbose=verbose,
                                       callbacks = [earlystop])

        if save_name:
            self.model.save(save_name)
            np.save('history_' + save_name +'.npy', self.history.history)


    def learning_curves(self):
```

```
        history = self.history
        epoches = len(history.history["loss"])
        mse = history.history["mean_squared_error"]
        val_mse = history.history["val_mean_squared_error"]

        loss = history.history["loss"]
        val_loss = history.history["val_loss"]

        epochs_range = range(epoches)

        fig = plt.figure(figsize=(12,6))

        plt.subplot(1,2,1)
        plt.plot(epochs_range, mse, label="train mse")
        plt.plot(epochs_range, val_mse, label="validataion mse")
        plt.title("mse")
        plt.xlabel("Epoch")
        plt.ylabel("mse")
        plt.legend(loc="lower right")

        plt.subplot(1,2,2)
        plt.plot(epochs_range, loss, label="train loss")
        plt.plot(epochs_range, val_loss, label="validataion loss")
        plt.title("Loss")
        plt.xlabel("Epoch")
        plt.ylabel("Loss")
        plt.legend(loc="upper right")

        fig.tight_layout()
        plt.show()


    def predict(self, X_test):
        load_name = self.params['load_name']
        if load_name:
            m_loaded = load_model("./"+load_name)
            pred_test = m_loaded.predict(X_test)
        else:
            pred_test = self.model.predict(X_test)

        return pred_test


    def show_result(self, pl, img_train, y_train, y_pred, best=True):
        input_shape = self.params['input_shape']
        each_loss = [MeanSquaredError()(y_train[i], y_pred[i]).numpy() for i in␣
↪range(y_pred.shape[0])]
```

```
        if best:
            index = np.argsort(each_loss)
        else:
            index = np.argsort(each_loss)[::-1]

        fig, axs= plt.subplots(1, pl, figsize = (80, 80*pl))
        for i in range(pl):
            axs[i].imshow(img_train[index[i]])
            x1, y1, x2, y2 = y_train[index[i]]*input_shape[0]
            axs[i].
↪add_patch(Rectangle((x1,y1),x2-x1,y2-y1,linewidth=5,edgecolor='b',facecolor='none'))
            x1, y1, x2, y2 = y_pred[index[i]]*input_shape[0]
            axs[i].
↪add_patch(Rectangle((x1,y1),x2-x1,y2-y1,linewidth=5,edgecolor='r',facecolor='none'))
            axs[i].axis("off")

        plt.show()
```

2.6 Model Evaluation

Ingredients for training Define the params of the vgg model class. Specifing "imagenet" for 'weights' of the vgg model is the ***3rd memory control approach***

[11]:
```
params = {# params for vgg
          'input_shape':INPUT_SHAPE,
          'weights':"imagenet",
          'include_top':False,

          # params for model compile
          'optimizer':"adam",
          'loss': "mse", #
          'metrics':"MeanSquaredError()",
          'monitor':"mean_squared_error",

          'save_name':False,  # if no need to save, fill in 'False'
          'batch_size':4,
          'epochs':20,

          # for predict
          'load_name':False   # if no need to load, fill in 'False'
}
```

Load the images for training and validation with fuction load_resize_img.

[12]:
```
X_train, y_train = load_resize_img(y_dict, X_train, path=PATH,␣
↪image_size=IMG_SIZE, plot=False)
```

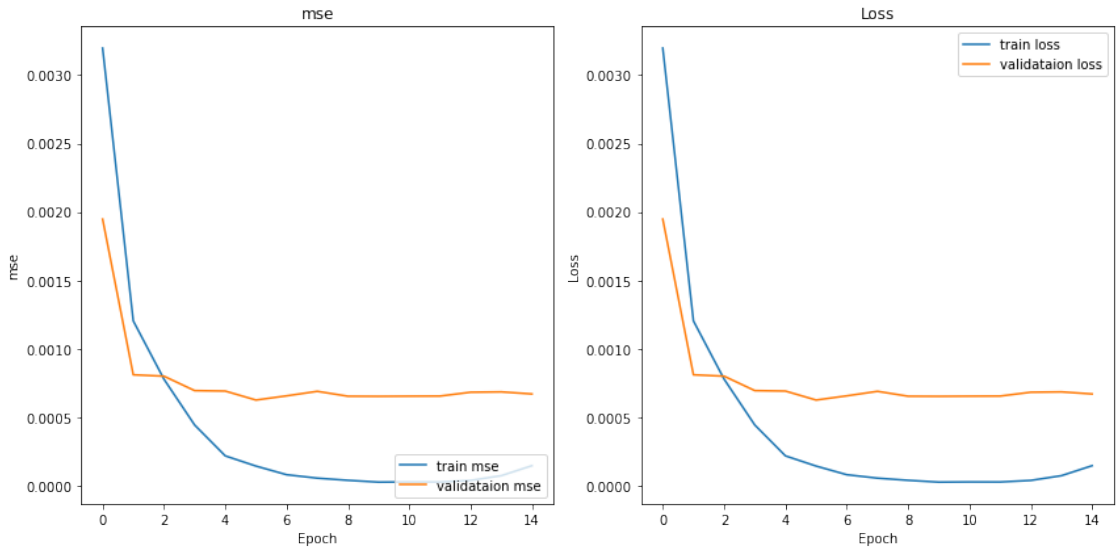
```
X_val, y_val = load_resize_img(y_dict, X_val, path=PATH, image_size=IMG_SIZE,␣
 ↪plot=False)
```

Since the task here is in regression flavour, we use mean squared error as our main metrics.

[13]:
```
vgg = VGG_detection(params, MUTE).get_vgg()
vgg.fit_vgg(X_train, X_val, y_train, y_val)
vgg.learning_curves()
```

```
2022-05-29 15:11:43.540580: I tensorflow/core/platform/cpu_feature_guard.cc:151]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.

Epoch 1/20
67/67 [==============================] - 26s 376ms/step - loss: 0.0032 -
mean_squared_error: 0.0032 - val_loss: 0.0019 - val_mean_squared_error: 0.0019
Epoch 2/20
67/67 [==============================] - 26s 394ms/step - loss: 0.0012 -
mean_squared_error: 0.0012 - val_loss: 8.1063e-04 - val_mean_squared_error:
8.1063e-04
Epoch 3/20
67/67 [==============================] - 25s 377ms/step - loss: 7.7887e-04 -
mean_squared_error: 7.7887e-04 - val_loss: 8.0063e-04 - val_mean_squared_error:
8.0063e-04
Epoch 4/20
67/67 [==============================] - 27s 404ms/step - loss: 4.4421e-04 -
mean_squared_error: 4.4421e-04 - val_loss: 6.9520e-04 - val_mean_squared_error:
6.9520e-04
Epoch 5/20
67/67 [==============================] - 24s 353ms/step - loss: 2.1856e-04 -
mean_squared_error: 2.1856e-04 - val_loss: 6.9180e-04 - val_mean_squared_error:
6.9180e-04
Epoch 6/20
67/67 [==============================] - 24s 365ms/step - loss: 1.4401e-04 -
mean_squared_error: 1.4401e-04 - val_loss: 6.2629e-04 - val_mean_squared_error:
6.2629e-04
Epoch 7/20
67/67 [==============================] - 26s 384ms/step - loss: 8.0896e-05 -
mean_squared_error: 8.0896e-05 - val_loss: 6.5699e-04 - val_mean_squared_error:
6.5699e-04
Epoch 8/20
67/67 [==============================] - 25s 370ms/step - loss: 5.5805e-05 -
mean_squared_error: 5.5805e-05 - val_loss: 6.8996e-04 - val_mean_squared_error:
6.8996e-04
Epoch 9/20
67/67 [==============================] - 30s 457ms/step - loss: 3.9965e-05 -
```

```
mean_squared_error: 3.9965e-05 - val_loss: 6.5420e-04 - val_mean_squared_error:
6.5420e-04
Epoch 10/20
67/67 [==============================] - 29s 431ms/step - loss: 2.6762e-05 -
mean_squared_error: 2.6762e-05 - val_loss: 6.5343e-04 - val_mean_squared_error:
6.5343e-04
Epoch 11/20
67/67 [==============================] - 25s 366ms/step - loss: 2.8201e-05 -
mean_squared_error: 2.8201e-05 - val_loss: 6.5454e-04 - val_mean_squared_error:
6.5454e-04
Epoch 12/20
67/67 [==============================] - 24s 353ms/step - loss: 2.7819e-05 -
mean_squared_error: 2.7819e-05 - val_loss: 6.5528e-04 - val_mean_squared_error:
6.5528e-04
Epoch 13/20
67/67 [==============================] - 25s 370ms/step - loss: 3.9201e-05 -
mean_squared_error: 3.9201e-05 - val_loss: 6.8338e-04 - val_mean_squared_error:
6.8338e-04
Epoch 14/20
67/67 [==============================] - 24s 365ms/step - loss: 7.3815e-05 -
mean_squared_error: 7.3815e-05 - val_loss: 6.8584e-04 - val_mean_squared_error:
6.8584e-04
Epoch 15/20
67/67 [==============================] - 24s 361ms/step - loss: 1.4680e-04 -
mean_squared_error: 1.4680e-04 - val_loss: 6.7069e-04 - val_mean_squared_error:
6.7069e-04
```

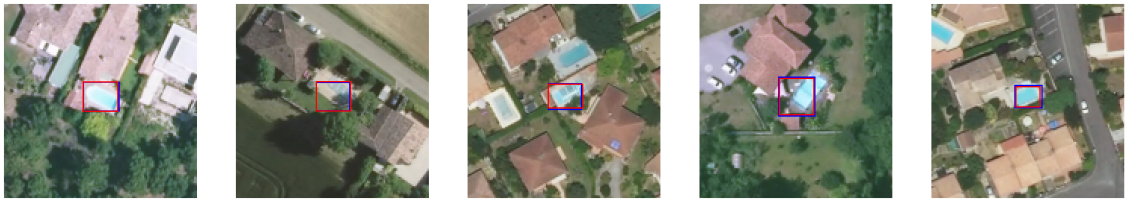


2.7 Prediction

In this section, we predict bounding box on test images and show the top 5 best and worst images with predicted box (in red) and true box (in blue).

[14]:
```
l_xtrain = [X_train]
l_ytrain = [y_train]
l_xval = [X_val]
l_yval = [y_val]
del X_train, y_train, X_val, y_val
del l_xtrain, l_ytrain, l_xval, l_yval
```

[15]:
```
X_test, y_test = load_resize_img(y_dict, X_test, path=PATH, image_size=IMG_SIZE,␣
↪plot=False)
y_pred = vgg.predict(X_test)
vgg.show_result(5, X_test, y_test, y_pred, best=True)
```



The blue polygons are the original boundaries while the red bounding boxes are the bounding boxes we predicted using our model. We see that overall our model was able to predict the location of the swimming pools. The best predictions are almost identical to the true boundaries, while for the worst ones, predicted bounding boxes are far from their true targets. Because some pools don't have a clear color, which makes the model difficult to differentiate them from the surroundings (from the 1st 3rd and 4th worst predictions below).

[16]:
```
vgg.show_result(5, X_test, y_test, y_pred, best=False)
```

The last code shows a summary of the model.

[17]:
```
vgg.model.summary()
```

```
Model: "model"
_________________________________________________________________
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 128, 128, 3)]	0
block1_conv1 (Conv2D)	(None, 128, 128, 64)	1792
block1_conv2 (Conv2D)	(None, 128, 128, 64)	36928
block1_pool (MaxPooling2D)	(None, 64, 64, 64)	0
block2_conv1 (Conv2D)	(None, 64, 64, 128)	73856
block2_conv2 (Conv2D)	(None, 64, 64, 128)	147584
block2_pool (MaxPooling2D)	(None, 32, 32, 128)	0
block3_conv1 (Conv2D)	(None, 32, 32, 256)	295168
block3_conv2 (Conv2D)	(None, 32, 32, 256)	590080
block3_conv3 (Conv2D)	(None, 32, 32, 256)	590080
block3_pool (MaxPooling2D)	(None, 16, 16, 256)	0
block4_conv1 (Conv2D)	(None, 16, 16, 512)	1180160
block4_conv2 (Conv2D)	(None, 16, 16, 512)	2359808
block4_conv3 (Conv2D)	(None, 16, 16, 512)	2359808
block4_pool (MaxPooling2D)	(None, 8, 8, 512)	0
block5_conv1 (Conv2D)	(None, 8, 8, 512)	2359808
block5_conv2 (Conv2D)	(None, 8, 8, 512)	2359808
block5_conv3 (Conv2D)	(None, 8, 8, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 128)	1048704
dense_1 (Dense)	(None, 64)	8256
dense_2 (Dense)	(None, 32)	2080

```
 dense_3 (Dense)             (None, 4)                 132

=================================================================
Total params: 15,773,860
Trainable params: 1,059,172
Non-trainable params: 14,714,688
_________________________________________________________________
```

3 Assignment 3

3.1 Introduction

For this assignment we created a comment-prediction model based on the comments from two Twitch streamers: PROD and Jinnytty. Both of which are very active and popular streamers and therefore have active chats. The goal is to tell, for any random comment in the chat, whether it belongs to PROD's or Jinnytty's chat. Since PROD is a Valorant streamer and Jinnytty usually does real-life travel streams, the structure of the chats should be completely different. Prediction should be relatively easy to make. However, going in we suspect difficulties with popular Twitch/streaming words like "Kek" or "Lul". We will attempt to encode messages while retaining context through word2vec and then do a simple logistic regression classification.

3.2 Saving stream data to local machine

We start by importing the Spark framework into the notebook:

[1]:
```
import threading

# Helper thread to avoid the Spark StreamingContext from blocking Jupyter

class StreamingThread(threading.Thread):
    def __init__(self, ssc):
        super().__init__()
        self.ssc = ssc
    def run(self):
        self.ssc.start()
        self.ssc.awaitTermination()
    def stop(self):
        print('----- Stopping... this may take a few seconds -----')
        self.ssc.stop(stopSparkContext=False, stopGraceFully=True)
```

[2]:
```
sc
```

[3]:
```
from pyspark.streaming import StreamingContext
```

Now we make a streaming thread and save the RDD files to a specified PATH variable.

[4]:
```
ssc = StreamingContext(sc, 30)
```

[5]:
```
lines = ssc.socketTextStream("localhost", 8080)
PATH = ...
lines.saveAsTextFiles(PATH)
```

[6]:
```
ssc_t = StreamingThread(ssc)
```

[7]:
```
ssc_t.start()
```

[8]:
```
ssc_t.stop()
```

```
----- Stopping... this may take a few seconds -----
```

3.3 Reading in data and training model

Usually one would work in the following way: through streamed RDD's one can construct a data frame through SparkSQL. This is easily done through .union() operations and any necessary extra columns based on other columns can be made through either .withColumn() or .select(). However, it is reported online that using .withColumn() is not advised as it results in slow data frame accessing. Another way of doing this is by having the RDD's converted into a Pandas data frame which can be easily converted into a SparkSQL data frame at any point. We opted to do this for our model training data as it resulted in much faster computation than when we built the data frame out of RDDs. Constructing the necessary columns this way takes much less code as well.

[5]:
```
import pandas as pd
import pyspark.sql.functions as F
import string
```

In the following code we construct a data frame by converting each saved RDD from the two streams into a Pandas data frame and then concatenating them one by one. We construct two new columns: messagesplit and label. Messagesplit will contain the message strings set to lower case, split by whitespace and stripped from special characters. We chose not to remove stop words as these chat messages are by nature very short and full of spelling mistakes. We suspect it would not benefit the cause at all. Label will have a float label (1.0 or 0.0) signifying whether the message came from Jinnytty's chat or not. In the end we create the SparkSQL data frame from the Pandas one.

[6]:
```
## Pandas version

df=pd.DataFrame([])

for path, dir, files in os.walk(PATH):
    for file in files:
        if file.startswith('part'):
            subsubfiles = os.path.join(path,file)
            read=spark.read.json(subsubfiles)
            pandasDF=read.toPandas()
            df = pd.concat([df,pandasDF])

df["messagesplit"] = df["message"].str.lower()
df["messagesplit"] = df["messagesplit"].str.split(' ')
df["label"] = (df["channel"] == "#jinnytty").astype(float)

#take out punctuation
table = str.maketrans('', '', string.punctuation)
df_messagesplit2 = list()
for comment in list(df["messagesplit"]):
    stripped = [w.translate(table) for w in comment]
    df_messagesplit2.append(stripped)
```

```
print(df_messagesplit2)
df["messagesplit"] = df_messagesplit2


from pyspark.sql import SQLContext, SparkSession
sc2 = SparkSession.builder.getOrCreate()

spark_df = sc2.createDataFrame(df)
```

The following output shows how many spelling errors there are, but also how unique the choice of words are from the respective fans. Some chat messages are clearly from a certain streamer. The word "crosshair" is a shooter term and is therefore directly linked to PROD. Emotes like ":yyjpopcorn" are used by Jinnytty subscribers. Sometimes the streamer's name pops up too, e.g. "jinny" in the second-to-last line.

```
[['squid1', 'trihard', 'squid2', 'squid4'], ['chew', 'jet', 'pek', '', 'hello'],
['is', 'super', 'sstrong'], ['yep'], ['kekw'], ['kekw'], ['dead', 'end'],
['kekw'], ['kekw'], ['waytoodank'], ['lol'], ['lmaoooo'], ['yyjpopcorn',
'\U000e0000'], ['yep', 'its', 'annoying'], ['xqcomega', 'xqcomega'],
['kekwiggle'], ['dinkdonk'], ['pogging'], ['wonder', 'if', 'ppl', 'here',
'main', 'job', 'is', 'fishing'], ['kekw'], ['pepega'], ['deadend'], ['dont',
'loook', 'behind', 'you', 'runnnn'], ['kekw'], ['esp', 'at', 'night'],
['farisft', 'bot', 'frag'], ['yeah', 'ofc'], ['kappa'], ['kappa'], ['teng',
'teng', 'teng', 'teng', 'teng', 'lul'], ['you', 'can', 'just', 'take', 'it',
'off', 'then'], ['lul'], ['kekw'], ['vincen202detect'], ['hmm'], ['res'],
['1920', 'x', '1080'], ['some1', 'house', 'eh'], ['what', 'if', 'its', 'make',
'a', 'sound', 'but', 'no', 'wind', 'monkaw'], ['you', 'can', 'just', 'put', 'a',
'rubber', 'band', 'on', 'it', 'nodders'], ['yokaza9595', 'wtf'], ['itsatko',
'he', 'doesnt', 'give', 'two', 'fucks', 'mate'], ['chet'], ['pepodetect'],
['we', 'pay', 'more', 'tax', 'of', 'own', 'pocket', 'for', 'the', 'others'],
['bruv'], ['crosshair'], ['pogu'], ['wowers'], ['pepodetect'], ['where', 'are',
'you', 'headed', 'jinny'], ['pog'], ['looks', 'like', 'it'], ['kekyou', 'thats',
'someone', 'house'], ['dead', 'end', 'monkaw'], ['yes'], ['let', 'go'],...]
```

We split the set into a training and test set and set the seed to four for reproducibility.

[7]:
```
training, test = spark_df.randomSplit([0.8,0.2],seed=4)
```

As we would also like to capture the context around certain keywords like the ones we saw above, we decided to encode this through word2vec. A more robust algorithm like Meta's Misspelling Oblivious Embeddings would probably have improved the result quite a bit, but that kind of technology is not available for SparkML.

Due to the many misspellings we set the word2vec vector size to 300 (three times more than the default size) so that we can give enough room to words and their popular misspellings. This might not have been needed, but there is no easy way to figure that out. The minimum count for a word to be included in word2vec is set to five. To make our lives a little bit easier, we one-hot-encoded the usernames present in the chat to test them on the prediction later. However, this turned out to be too powerful as the fan bases of both streamers are very loyal and almost always return. Most of the prediction power will be due to this addition. In the following code you can see the setup

of the pipeline for the data pre-processing and training with logistic regression.

[8]:
```
from pyspark.ml.feature import Word2Vec
word2Vec = Word2Vec(vectorSize=300,minCount=5,inputCol="messagesplit", #end up␣
 ↪with a vector of size 300
                    outputCol="result")                                  #minimum␣
 ↪count of a word to be included in word2vec = 5
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import (
    VectorAssembler,
    OneHotEncoder,
    StringIndexer
)

stringIndexer =␣
 ↪StringIndexer(inputCol="username",outputCol="username_StringIndexer",
                handleInvalid="skip")

ohe = OneHotEncoder(inputCols=["username_StringIndexer"],
                    outputCols=["username_OneHotEncoder"])

# in spark you need to transform all features into one feature vector
assemblerInput = ["result"]
assemblerInput += ["username_OneHotEncoder"]

vector_assembler = VectorAssembler(inputCols = assemblerInput, outputCol =␣
 ↪"VectorAssembler_Features")

#Now we assemble the pipeline
stages = []
stages += [word2Vec]
stages += [stringIndexer]
stages += [ohe]
stages += [vector_assembler]

from pyspark.ml import Pipeline

pipeline = Pipeline().setStages(stages)

#pipeline is created (above), it is fitted on the training data(below)

model = pipeline.fit(training)
training_df = model.transform(training)

#word2vec ends, logreg starts
```

```
import pyspark.sql.functions as F
data = training_df.select(F.col("VectorAssembler_Features").alias("features"),␣
 ↪#all features need to be in one column and we need to call that column␣
 ↪"features"
                          F.col("label"))

lrModel = LogisticRegression().fit(data)
```

Because adding the usernames of the fans to the prediction model is so powerful, we also train a model purely on the word2vec column:

[25]:
```
# in spark you need to transform all features into one feature vector
assemblerInput2 = ["result"]
vector_assembler2 = VectorAssembler(inputCols = assemblerInput2, outputCol =␣
 ↪"VectorAssembler_Features")

#Now we assemble the pipeline
stages2 = []
stages2 += [word2Vec]
stages2 += [vector_assembler2]

pipeline2 = Pipeline().setStages(stages2)

#pipeline is created (above), it is fitted on the training data(below)

model2 = pipeline2.fit(training)
training_df2 = model2.transform(training)

#word2vec ends, logreg starts

data2 = training_df2.select(F.col("VectorAssembler_Features").alias("features"),␣
 ↪#all features need to be in one column and we need to call that column␣
 ↪"features"
                          F.col("label"))

lrModel2 = LogisticRegression().fit(data2)
```

3.4 Deploying the model on stream

We decided to deploy the model above on both streams two days later. We suspected there to be other fans, but as we will soon discover, this was not the case.

[44]:
```
from pyspark.sql import Row
from pyspark.sql.functions import udf, struct, array, col, lit
from pyspark.sql.types import StringType
from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.sql.types import FloatType
```

The operations we applied to the training data, we also apply on the streamed data. Again, we could have done this through SparkSQL (and we did), but the code would have been too slow and cumbersome to display. Note that the process function also includes code to print a confusion matrix.

[19]:
```
def process(time, rdd):
    if rdd.isEmpty():
        return

    print("========= %s =========" % str(time))
    # Convert to data frame
    df = spark.read.json(rdd)
    df = df.toPandas()

    df["messagesplit"] = df["message"].str.lower()

    df["messagesplit"] = df["messagesplit"].str.split(' ')
    df["label"] = (df["channel"] == "#jinnytty").astype(float)
    table = str.maketrans('', '', string.punctuation)
    df_messagesplit2 = list()
    for comment in list(df["messagesplit"]):
        stripped = [w.translate(table) for w in comment]
        df_messagesplit2.append(stripped)
    df["messagesplit"] = df_messagesplit2

    sc_pro = SparkSession.builder.getOrCreate()
    spark_df_pro = sc_pro.createDataFrame(df)

    #trained model is applied to the streamed data
    pp_df = model.transform(spark_df_pro)
    #pp_df.show()
    testData = pp_df.select(F.col("VectorAssembler_Features").
↪alias("features"),F.col("label"))


    predictions = lrModel.transform(testData)
    preds_and_labels=predictions.select(F.col("label").cast(FloatType()),F.
↪col("prediction"))
    preds_and_labels.show()
    preds_and_labels = preds_and_labels.select(['prediction','label'])
    metrics = MulticlassMetrics(preds_and_labels.rdd.map(tuple))
    print(metrics.confusionMatrix().toArray())
```

Again, because we want to see the model work without usernames, we create a process based on the model trained purely on the word2vec column:

[45]:
```
def process2(time, rdd):
    if rdd.isEmpty():
```

```
        return

    print("========= %s =========" % str(time))
    # Convert to data frame
    df = spark.read.json(rdd)
    df = df.toPandas()

    df["messagesplit"] = df["message"].str.lower()

    df["messagesplit"] = df["messagesplit"].str.split(' ')
    df["label"] = (df["channel"] == "#jinnytty").astype(float)
    table = str.maketrans('', '', string.punctuation)
    df_messagesplit2 = list()
    for comment in list(df["messagesplit"]):
        stripped = [w.translate(table) for w in comment]
        df_messagesplit2.append(stripped)

    df["messagesplit"] = df_messagesplit2

    sc_pro = SparkSession.builder.getOrCreate()
    spark_df_pro = sc_pro.createDataFrame(df)



    #trained model is applied to the streamed data
    pp_df = model2.transform(spark_df_pro)
    #pp_df.show()
    testData = pp_df.select(F.col("VectorAssembler_Features").
↪alias("features"),F.col("label"))

    predictions = lrModel2.transform(testData)
    preds_and_labels=predictions.select(F.col("label").cast(FloatType()),F.
↪col("prediction"))
    preds_and_labels.show()
    preds_and_labels = preds_and_labels.select(['prediction','label'])
    metrics = MulticlassMetrics(preds_and_labels.rdd.map(tuple))
    print(metrics.confusionMatrix().toArray())
```

Starting the streaming thread:

[20]:
```
ssc = StreamingContext(sc, 10)
```

[21]:
```
lines = ssc.socketTextStream("localhost", 8080)
lines.foreachRDD(process)
```

[22]:
```
ssc_t = StreamingThread(ssc)
```

When the thread starts, the labels and their respective predictions are going to print. After each

of those, a confusion matrix will print to show how much were predicted right (a good predictive model will have low off-diagonal entries).

3.4.1 Prediction with usernames

[23]:
```
ssc_t.start()
```

```
========= 2022-05-24 13:52:40 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
+-----+----------+
```

When there was only one label present and the prediction was completely correct, an output is displayed as follows:

```
[[4.]]
========= 2022-05-24 13:52:50 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[16.]]
========= 2022-05-24 13:53:00 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[1. 0.]
 [0. 9.]]
========= 2022-05-24 13:53:10 =========
+-----+----------+
```

```
|label|prediction|
+-----+----------+
|  0.0|       0.0|
|  0.0|       0.0|
+-----+----------+

[[2.]]
========= 2022-05-24 13:53:20 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 9.  0.]
 [ 0. 31.]]
========= 2022-05-24 13:53:30 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       0.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 3.  0.]
 [ 0. 17.]]
========= 2022-05-24 13:53:40 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[15.]]
========= 2022-05-24 13:53:50 =========
```

```
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[5.]]
========= 2022-05-24 13:54:00 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[3. 0.]
 [0. 7.]]
========= 2022-05-24 13:54:10 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[2. 0.]
 [0. 7.]]
========= 2022-05-24 13:54:20 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
```

```
+-----+----------+
[[ 3.  0.]
 [ 0. 11.]]
```

We see that the predictions are always 100% right.

[24]:
```
ssc_t.stop()
```

```
----- Stopping... this may take a few seconds -----
========= 2022-05-24 13:54:30 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 2.  0.]
 [ 0. 13.]]
========= 2022-05-24 13:54:40 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[1. 0.]
 [0. 6.]]
========= 2022-05-24 13:54:50 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  0.0|       0.0|
|    ⋮|         ⋮|
+-----+----------+

[[1. 0.]
```

```
 [0. 4.]]
========= 2022-05-24 13:55:00 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 2.  0.]
 [ 0. 17.]]
========= 2022-05-24 13:55:10 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 2.  0.]
 [ 0. 10.]]
========= 2022-05-24 13:55:20 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 3.  0.]
 [ 0. 10.]]
========= 2022-05-24 13:55:30 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
```

```
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 2.  0.]
 [ 0. 16.]]
========= 2022-05-24 13:55:40 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  0.0|       0.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 3.  0.]
 [ 0. 20.]]
========= 2022-05-24 13:55:50 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 1.  0.]
 [ 0. 12.]]
========= 2022-05-24 13:56:00 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 1.  0.]
 [ 0. 23.]]
```

```
========= 2022-05-24 13:56:10 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 1.  0.]
 [ 0. 10.]]
========= 2022-05-24 13:56:20 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  0.0|       0.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 3.  0.]
 [ 0. 20.]]
========= 2022-05-24 13:56:30 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  0.0|       0.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 3.  0.]
 [ 0. 20.]]
========= 2022-05-24 13:56:40 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  0.0|       0.0|
|  1.0|       1.0|
```

```
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 1.  0.]
 [ 0. 17.]]
========= 2022-05-24 13:56:50 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[5.]]
========= 2022-05-24 13:57:00 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  0.0|       0.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 8.  0.]
 [ 0. 24.]]
```

3.4.2 Prediction without usernames

When we deploy the model not involving the username encodings, we get the following:

[47]:
```
ssc2 = StreamingContext(sc, 10)
```

[48]:
```
lines2 = ssc2.socketTextStream("localhost", 8080)
lines2.foreachRDD(process2)
```

[49]:
```
ssc_t2 = StreamingThread(ssc2)
```

[50]:
```
ssc_t2.start()
```

```
========= 2022-05-24 14:12:20 =========
+-----+----------+
```

```
|label|prediction|
+-----+----------+
|  0.0|       0.0|
|  1.0|       0.0|
|  1.0|       1.0|
|  1.0|       0.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 3.  5.]
 [ 4. 10.]]
========= 2022-05-24 14:12:30 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       1.0|
|  0.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[4. 2.]
 [0. 4.]]
========= 2022-05-24 14:12:40 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  0.0|       1.0|
|  0.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[2. 9.]
 [2. 9.]]
========= 2022-05-24 14:12:50 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       0.0|
|  0.0|       0.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
```

```
+-----+----------+

[[ 7.  4.]
 [ 2. 19.]]
========= 2022-05-24 14:13:00 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  0.0|       1.0|
|  1.0|       1.0|
|  1.0|       0.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 3. 15.]
 [ 6. 19.]]
========= 2022-05-24 14:13:10 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       1.0|
|  1.0|       0.0|
|    ⋮|         ⋮|
+-----+----------+

[[10. 18.]
 [ 1. 10.]]
========= 2022-05-24 14:13:20 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       1.0|
|  1.0|       1.0|
|  0.0|       0.0|
|  0.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[15. 10.]
 [ 0. 11.]]
========= 2022-05-24 14:13:30 =========
```

[51]:
```
ssc_t2.stop()
```

```
----- Stopping... this may take a few seconds -----
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       0.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  0.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 7. 10.]
 [ 1.  8.]]
========= 2022-05-24 14:13:40 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       0.0|
|  0.0|       1.0|
|  0.0|       0.0|
|  1.0|       0.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 5.  2.]
 [ 9. 14.]]
========= 2022-05-24 14:13:50 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 2.  1.]
 [ 2. 13.]]
========= 2022-05-24 14:14:00 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  0.0|       1.0|
```

```
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[3. 6.]
 [1. 9.]]
========= 2022-05-24 14:14:10 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       0.0|
|    ⋮|         ⋮|
+-----+----------+

[[1. 5.]
 [8. 7.]]
========= 2022-05-24 14:14:20 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       1.0|
|  1.0|       1.0|
|  0.0|       1.0|
|  1.0|       0.0|
|    ⋮|         ⋮|
+-----+----------+

[[3. 6.]
 [3. 6.]]
========= 2022-05-24 14:14:30 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       0.0|
|  0.0|       0.0|
|  0.0|       1.0|
|  0.0|       0.0|
|    ⋮|         ⋮|
+-----+----------+

[[8. 5.]
 [5. 3.]]
========= 2022-05-24 14:14:40 =========
```

```
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       0.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  0.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 2.  5.]
 [ 8. 17.]]
========= 2022-05-24 14:14:50 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       0.0|
|  0.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 2.  5.]
 [ 4. 18.]]
========= 2022-05-24 14:15:00 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  0.0|       0.0|
|    ⋮|         ⋮|
+-----+----------+

[[ 4.  2.]
 [ 3. 21.]]
========= 2022-05-24 14:15:10 =========
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       1.0|
|  0.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
```

```
|    ⋮|         ⋮|
+-----+----------+

[[ 0.  4.]
 [ 0. 10.]]
```

The performance in general is amazing at times, but disappointing at others. Next section we will find out why precisely that is.

3.5 Test set performance evaluation

It could be interesting to see why exactly the prediction fails at times. To find out we transform the test data display the messages when the prediction is right and when it is wrong.

[45]:
```
test_pred_df = model2.transform(test)
```

[46]:
```
testdata = test_pred_df.select(F.col("VectorAssembler_Features").
 ↪alias("features"), #all features need to be in one column and we need to call␣
 ↪that column "features"
                          F.col("label"))

#logistic regression is fitted on the transformed data
predictions = lrModel2.transform(testdata)
```

[47]:
```
from pyspark.sql.window import Window
from pyspark.sql.functions import monotonically_increasing_id, row_number

w = Window.orderBy(monotonically_increasing_id())

#join message column with prediction and label by row number
preds_labels=  predictions.select(F.col("label").cast(FloatType()),F.
 ↪col("prediction")).withColumn("columnindex", row_number().over(w))
message=  test.select("message").withColumn("columnindex", row_number().over(w))

preds_labels_message = preds_labels.join(message, preds_labels.columnindex ==␣
 ↪message.columnindex, 'inner').drop(message.columnindex)
preds_labels_message.show()
```

```
+-----+----------+-----------+--------------------+
|label|prediction|columnindex|             message|
+-----+----------+-----------+--------------------+
|  1.0|       1.0|          1|Squid1 TriHard Sq...|
|  1.0|       1.0|          2|          WAYTOODANK|
|  1.0|       1.0|          3|wonder if ppl her...|
|  1.0|       1.0|          4|        esp at night|
|  1.0|       1.0|          5|                 Lul|
|  1.0|       1.0|          6|     vincen202Detect|
|  1.0|       0.0|          7|                CHET|
```

```
|  1.0|       1.0|          8|          PepoDetect|
|  1.0|       1.0|          9|we pay more tax o...|
|  1.0|       1.0|         10|                 Pog|
|  1.0|       1.0|         11|              let go|
|  1.0|       1.0|         12|                PogU|
|  1.0|       1.0|         13|          yyjPopcorn|
|  1.0|       0.0|         14|are these people'...|
|  1.0|       1.0|         15|          looks dope|
|  1.0|       1.0|         16|          super lost|
|  1.0|       1.0|         17|                lost|
|  1.0|       1.0|         18|        you  are YEP|
|  1.0|       1.0|         19|     better Thailand|
|  1.0|       1.0|         20|            jessuSus|
+-----+----------+-----------+--------------------+
only showing top 20 rows
```

[48]:
```
#label = 1, prediction = 1
preds_labels_message.filter(preds_labels_message.label+preds_labels_message.
 ↪prediction>1).collect()
```

[48]:
```
[Row(label=1.0, prediction=1.0, columnindex=1, message='Squid1 TriHard Squid2
Squid4'),
 Row(label=1.0, prediction=1.0, columnindex=2, message='WAYTOODANK'),
 Row(label=1.0, prediction=1.0, columnindex=3, message='wonder if ppl here main
job is fishing'),
 Row(label=1.0, prediction=1.0, columnindex=4, message='esp at night'),
 Row(label=1.0, prediction=1.0, columnindex=5, message='Lul'),
 Row(label=1.0, prediction=1.0, columnindex=6, message='vincen202Detect'),
 Row(label=1.0, prediction=1.0, columnindex=8, message='PepoDetect'),
 Row(label=1.0, prediction=1.0, columnindex=9, message='we pay more tax of own
pocket for the others'),
 Row(label=1.0, prediction=1.0, columnindex=10, message='Pog'),
 Row(label=1.0, prediction=1.0, columnindex=11, message='let go'),
 Row(label=1.0, prediction=1.0, columnindex=12, message='PogU'),
 Row(label=1.0, prediction=1.0, columnindex=13, message='yyjPopcorn'),
 Row(label=1.0, prediction=1.0, columnindex=15, message='looks dope'),
 Row(label=1.0, prediction=1.0, columnindex=16, message='super lost'),
 Row(label=1.0, prediction=1.0, columnindex=17, message='lost'),
 Row(label=1.0, prediction=1.0, columnindex=18, message='you  are YEP'),
 Row(label=1.0, prediction=1.0, columnindex=19, message='better Thailand'),
 Row(label=1.0, prediction=1.0, columnindex=20, message='jessuSus'),
 Row(label=1.0, prediction=1.0, columnindex=21, message='pepeMeltdown'),
 Row(label=1.0, prediction=1.0, columnindex=23, message='maybe a boat'),
 Row(label=1.0, prediction=1.0, columnindex=24, message='yeah super LOST'),
 Row(label=1.0, prediction=1.0, columnindex=25, message='poop water PepeLaugh'),
 Row(label=1.0, prediction=1.0, columnindex=26, message='HUH'),
```

```
 Row(label=1.0, prediction=1.0, columnindex=27, message='YEP'),
 Row(label=1.0, prediction=1.0, columnindex=28, message='NODDERS 4'),
 Row(label=1.0, prediction=1.0, columnindex=29, message='KEKW'),
 Row(label=1.0, prediction=1.0, columnindex=30, message='KEKWalk'),
 Row(label=1.0, prediction=1.0, columnindex=31, message='Google semporna'),
 Row(label=1.0, prediction=1.0, columnindex=34, message='yyjPopcorn
\U000e0000'),
 Row(label=1.0, prediction=1.0, columnindex=35, message='maybe go through saba
to australia'),
 Row(label=1.0, prediction=1.0, columnindex=36, message='Sharks in 3.. 2..
1...'),
 Row(label=1.0, prediction=1.0, columnindex=37, message='yo!'),
 Row(label=1.0, prediction=1.0, columnindex=38, message='yep'),
 Row(label=1.0, prediction=1.0, columnindex=39, message='go Ipoh'),
 Row(label=1.0, prediction=1.0, columnindex=41, message='WHAT IF YOUR PHONE FALL
INTO THE HOLE'),
 Row(label=1.0, prediction=1.0, columnindex=42, message='YEP it was'),
 Row(label=1.0, prediction=1.0, columnindex=43, message='no livestream in
Sabah...'),
 Row(label=1.0, prediction=1.0, columnindex=44, message="animals aren't clean
so"),
 Row(label=1.0, prediction=1.0, columnindex=46, message='there is a lot of
island in Malaysia that has clean water'),
 Row(label=1.0, prediction=1.0, columnindex=47, message='PepeLaugh surely'),
 Row(label=1.0, prediction=1.0, columnindex=48, message='nice'),
 Row(label=1.0, prediction=1.0, columnindex=49, message='Nice'),
 Row(label=1.0, prediction=1.0, columnindex=50, message='nice'),
 Row(label=1.0, prediction=1.0, columnindex=51, message='Sunway Tambun mountain
guy'),
 Row(label=1.0, prediction=1.0, columnindex=52, message='EleGiggle "nice"'),
 Row(label=1.0, prediction=1.0, columnindex=53, message='WHAT IF YOUR PHONE FALL
INTO THE HOL.E'),
 Row(label=1.0, prediction=1.0, columnindex=54, message='like 110%'),
 Row(label=1.0, prediction=1.0, columnindex=55, message='AYAYA'),
 Row(label=1.0, prediction=1.0, columnindex=56, message='EZ'),
 Row(label=1.0, prediction=1.0, columnindex=57, message='EZ Clap'),
 Row(label=1.0, prediction=1.0, columnindex=58, message='HUH'),
 Row(label=1.0, prediction=1.0, columnindex=59, message='AYAYA'),
 Row(label=1.0, prediction=1.0, columnindex=60, message='EZ'),
 Row(label=1.0, prediction=1.0, columnindex=61, message='EZ'),
 Row(label=1.0, prediction=1.0, columnindex=62, message='USED BAGS'),
 Row(label=1.0, prediction=1.0, columnindex=63, message='TEMPLE'),
 Row(label=1.0, prediction=1.0, columnindex=64, message='EZ'),
 Row(label=1.0, prediction=1.0, columnindex=65, message='EZ Clap'),
 Row(label=1.0, prediction=1.0, columnindex=66, message='legend go to sleep'),
 Row(label=1.0, prediction=1.0, columnindex=67, message='yyj55dobal
yyj55dobal'),
```

```
 Row(label=1.0, prediction=1.0, columnindex=68, message='DansGame'),
 Row(label=1.0, prediction=1.0, columnindex=69, message='sandakan?'),
 Row(label=1.0, prediction=1.0, columnindex=70, message='I love onion'),
 Row(label=1.0, prediction=1.0, columnindex=71, message='HUH'),
 Row(label=1.0, prediction=1.0, columnindex=72, message='VoHiYo'),
 Row(label=1.0, prediction=1.0, columnindex=73, message='mmmmm, pooopy'),
 Row(label=1.0, prediction=1.0, columnindex=74, message='yyjPog yyjPog yyjPog'),
 Row(label=1.0, prediction=1.0, columnindex=75, message='Straits of Malacca, the
sea here is one of the busiest shipping routes in the world YEP'),
 Row(label=1.0, prediction=1.0, columnindex=76, message='wtf'),
 Row(label=1.0, prediction=1.0, columnindex=77, message='smell poop?'),
 Row(label=1.0, prediction=1.0, columnindex=78, message='DansChamp Brown
water'),
 Row(label=1.0, prediction=1.0, columnindex=79, message='yyjSmile yyjSmile'),
 Row(label=1.0, prediction=1.0, columnindex=80, message='yyjDisgust'),
 Row(label=1.0, prediction=1.0, columnindex=81, message='KEKWalk'),
 Row(label=1.0, prediction=1.0, columnindex=82, message='Y OOOO'),
```

[49]:
```
#label = 0, prediction = 0
preds_labels_message.filter(preds_labels_message.label+preds_labels_message.
 ↪prediction<1).collect()
```

[49]:
```
[Row(label=0.0, prediction=0.0, columnindex=132, message='LMAOOOO'),
 Row(label=0.0, prediction=0.0, columnindex=137, message='@nottooshabby2002 0.5
800dpi, 1.2 scoped'),
 Row(label=0.0, prediction=0.0, columnindex=138, message='!rank'),
 Row(label=0.0, prediction=0.0, columnindex=140, message='@unkn0wnaa IRON 1'),
 Row(label=0.0, prediction=0.0, columnindex=141, message='@PROD you need to solo
to immo with your ego so you can get humbled'),
 Row(label=0.0, prediction=0.0, columnindex=142, message='nt gg'),
 Row(label=0.0, prediction=0.0, columnindex=145, message='SO BAD'),
 Row(label=0.0, prediction=0.0, columnindex=146, message='lmaooo'),
 Row(label=0.0, prediction=0.0, columnindex=147, message='All u need is the
music buff'),
 Row(label=0.0, prediction=0.0, columnindex=150, message='yeh its the crosshair
COPIUM'),
 Row(label=0.0, prediction=0.0, columnindex=154, message='what res @PROD'),
 Row(label=0.0, prediction=0.0, columnindex=155, message='!RANK'),
 Row(label=0.0, prediction=0.0, columnindex=156, message='@ItsAtko ill prove it
to you if he dosent dodge me'),
 Row(label=0.0, prediction=0.0, columnindex=163, message='lmao'),
 Row(label=0.0, prediction=0.0, columnindex=164, message='u did a body tpp'),
 Row(label=0.0, prediction=0.0, columnindex=165, message='"I\'m doing bad this
match" like almost every match'),
 Row(label=0.0, prediction=0.0, columnindex=166, message='Bro why do u keep
```

```
dying'),
 Row(label=0.0, prediction=0.0, columnindex=173, message='play sum music?'),
 Row(label=0.0, prediction=0.0, columnindex=174, message='Thank god u playing
with emil it’s so funny'),
 Row(label=0.0, prediction=0.0, columnindex=176, message='THE MAP'),
 Row(label=0.0, prediction=0.0, columnindex=180, message='!rank'),
 Row(label=0.0, prediction=0.0, columnindex=181, message='gg'),
 Row(label=0.0, prediction=0.0, columnindex=183, message='u wish u carried
kid'),
 Row(label=0.0, prediction=0.0, columnindex=184, message='gg'),
 Row(label=0.0, prediction=0.0, columnindex=185, message='gg'),
 Row(label=0.0, prediction=0.0, columnindex=189, message='prod in a backpack'),
 Row(label=0.0, prediction=0.0, columnindex=191, message='14 26 lol'),
 Row(label=0.0, prediction=0.0, columnindex=195, message='ur bad @PROD'),
 Row(label=0.0, prediction=0.0, columnindex=197, message='@senpaipapi2170 MAIN:
Immortal 1 - 38 RR ; ALT: Immortal 2 - 132 RR'),
 Row(label=0.0, prediction=0.0, columnindex=198, message='bro im the same rank
as u'),
 Row(label=0.0, prediction=0.0, columnindex=204, message='@PROD dont dodge the
1v1 im chats represantator UR BAD PROD 14-26'),
 Row(label=0.0, prediction=0.0, columnindex=206, message='D:'),
 Row(label=0.0, prediction=0.0, columnindex=208, message='ummm actually im plat2
rn @PROD'),
 Row(label=0.0, prediction=0.0, columnindex=209, message='PROD washed'),
 Row(label=0.0, prediction=0.0, columnindex=211, message='14-26'),
 Row(label=0.0, prediction=0.0, columnindex=212, message="I'm bronze 3 actually),
 Row(label=0.0, prediction=0.0, columnindex=213, message='why so mad'),
 Row(label=0.0, prediction=0.0, columnindex=309, message='LMAOOO'),
 Row(label=0.0, prediction=0.0, columnindex=313, message='26 deaths HOLYYY
GIGACHAD'),
 Row(label=0.0, prediction=0.0, columnindex=320, message='he mad mad'),
 Row(label=0.0, prediction=0.0, columnindex=322, message='copium'),
 Row(label=0.0, prediction=0.0, columnindex=323, message='prod reusing the same
insult every stream LUL'),
 Row(label=0.0, prediction=0.0, columnindex=325, message='@jerrybenand MAIN:
Immortal 1 - 38 RR ; ALT: Immortal 2 - 132 RR'),
 Row(label=0.0, prediction=0.0, columnindex=327, message='CLASHROYALE GOD Xd'),
 Row(label=0.0, prediction=0.0, columnindex=329, message='he said 45'),
 Row(label=0.0, prediction=0.0, columnindex=331, message='14/26'),
 Row(label=0.0, prediction=0.0, columnindex=335, message='!sens'),
 Row(label=0.0, prediction=0.0, columnindex=336, message='true'),
 Row(label=0.0, prediction=0.0, columnindex=337, message='D:'),
 Row(label=0.0, prediction=0.0, columnindex=338, message='@PROD UR A CRY BABY
HOLYY'),
 Row(label=0.0, prediction=0.0, columnindex=339, message='BabyRage'),
 Row(label=0.0, prediction=0.0, columnindex=343, message='14/26'),
 Row(label=0.0, prediction=0.0, columnindex=346, message='BabyRage'),
```

```
 Row(label=0.0, prediction=0.0, columnindex=348, message='PROD IS RIGHT LMAO'),
 Row(label=0.0, prediction=0.0, columnindex=351, message='@PROD STOP
INSTALOCKING JETT UR SHIT'),
 Row(label=0.0, prediction=0.0, columnindex=352, message='D:'),
 Row(label=0.0, prediction=0.0, columnindex=353, message='BabyRage BabyRage'),
 Row(label=0.0, prediction=0.0, columnindex=354, message='LROD cringe'),
 Row(label=0.0, prediction=0.0, columnindex=356, message='PLEASE DONT PLAY
JETT'),
 Row(label=0.0, prediction=0.0, columnindex=357, message='D:'),
 Row(label=0.0, prediction=0.0, columnindex=360, message='@znowavy i mean
diamond 2'),
 Row(label=0.0, prediction=0.0, columnindex=362, message='BabyRage'),
 Row(label=0.0, prediction=0.0, columnindex=371, message='why he is so angry wtf
xDDD'),
 Row(label=0.0, prediction=0.0, columnindex=374, message='!sens'),
 Row(label=0.0, prediction=0.0, columnindex=375, message='emil loves u prod')...
```

It becomes immediately clear what the model thinks are PROD and Jinnytty type messages. Jinny's chat is filled with a lot of emotes like :yyjPopcorn, :yyj55dobal, :yyjPog, etc. But also with terms like pog, ez and kek. In general, because so many diverse things happen on Jinny's stream, it is easier to tell what messages come from a PROD chat; most of his messages contain words like rank, crosshair, res, body, bad, rage, mad, etc.

We will also take a look at wrongly predicted messages:

[50]:
```
#label = 1, prediction = 0
preds_labels_message.filter((preds_labels_message.label==1.0) &␣
 ↪(preds_labels_message.prediction==0.0)).collect()
```

[50]:
```
[Row(label=1.0, prediction=0.0, columnindex=7, message='CHET'),
 Row(label=1.0, prediction=0.0, columnindex=14, message="are these people's
homes?"),
 Row(label=1.0, prediction=0.0, columnindex=22, message='4Shrug'),
 Row(label=1.0, prediction=0.0, columnindex=32, message='Women comes out yelling
with a BROOM LUL LUL LUL'),
 Row(label=1.0, prediction=0.0, columnindex=33, message='sabah then'),
 Row(label=1.0, prediction=0.0, columnindex=40, message='Sabah need to climb
tree for internet no'),
 Row(label=1.0, prediction=0.0, columnindex=45, message='Enjoying the stream?
Subscribe to support the channel! https://www.twitch.tv/subs/jinnytty'),
 Row(label=1.0, prediction=0.0, columnindex=95, message='DONT SAY'),
 Row(label=1.0, prediction=0.0, columnindex=103, message='the toilets flush into
the ocean'),
 Row(label=1.0, prediction=0.0, columnindex=127, message='ye'),
 Row(label=1.0, prediction=0.0, columnindex=130, message='Check out the
temple?'),
 Row(label=1.0, prediction=0.0, columnindex=221, message='hi'),
```

```
 Row(label=1.0, prediction=0.0, columnindex=250, message='that is Hanuman the
HUMAN GOD'),
 Row(label=1.0, prediction=0.0, columnindex=273, message='Chinese have bad
gambling habits'),
 Row(label=1.0, prediction=0.0, columnindex=276, message='Oh'),
 Row(label=1.0, prediction=0.0, columnindex=278, message='???'),
 Row(label=1.0, prediction=0.0, columnindex=287, message='??'),
 Row(label=1.0, prediction=0.0, columnindex=306, message='dragon balls great,
muten roshi is bad TriHard TriHard TriHard TriHard TriHard RyuChamp RyuChamp
RyuChamp RyuChamp'),
 Row(label=1.0, prediction=0.0, columnindex=307, message='Only weebs'),
 Row(label=1.0, prediction=0.0, columnindex=421, message='dragonball literally
the most normie and popular anime'),
 Row(label=1.0, prediction=0.0, columnindex=425, message='Literally the main
character lol'),
 Row(label=1.0, prediction=0.0, columnindex=453, message='His name is Hanuman he
was a boy who challenged the GOds because he didnt believe death was
necessary'),
 Row(label=1.0, prediction=0.0, columnindex=461, message='is that water bottle
the running gag ?'),
 Row(label=1.0, prediction=0.0, columnindex=479, message='is it a boat?'),
 Row(label=1.0, prediction=0.0, columnindex=485, message='ok'),
 Row(label=1.0, prediction=0.0, columnindex=488, message='I need a fresh beer
man  TRUEING'),
 Row(label=1.0, prediction=0.0, columnindex=499, message='that is an alley'),
 Row(label=1.0, prediction=0.0, columnindex=501, message='Dont'),
 Row(label=1.0, prediction=0.0, columnindex=514, message='oh no tourist trap'),
 Row(label=1.0, prediction=0.0, columnindex=519, message='WADDLE'),
 Row(label=1.0, prediction=0.0, columnindex=524, message='ye'),
 Row(label=1.0, prediction=0.0, columnindex=651, message='peepoPopcorn'),
 Row(label=1.0, prediction=0.0, columnindex=652, message='IPOHIPOH THE BEST'),
 Row(label=1.0, prediction=0.0, columnindex=654, message='hi @Kaitypure
nokoWave1 nokoWave1'),
 Row(label=1.0, prediction=0.0, columnindex=667, message='lmao'),
 Row(label=1.0, prediction=0.0, columnindex=685, message='peepoPopcorn'),
 Row(label=1.0, prediction=0.0, columnindex=697, message='MOM DIESOFCRINGE'),
 Row(label=1.0, prediction=0.0, columnindex=705, message='what is Nasi Kandar'),
 Row(label=1.0, prediction=0.0, columnindex=710, message='jetty hehehe'),
 Row(label=1.0, prediction=0.0, columnindex=717, message='FeelsDankMan'),
 Row(label=1.0, prediction=0.0, columnindex=723, message='careful walking on the
street'),
 Row(label=1.0, prediction=0.0, columnindex=728, message='have u tried out memek
yet?'),
 Row(label=1.0, prediction=0.0, columnindex=734, message='ok'),
 Row(label=1.0, prediction=0.0, columnindex=756, message='Yoooo jetty'),
 Row(label=1.0, prediction=0.0, columnindex=876, message='what'),
 Row(label=1.0, prediction=0.0, columnindex=881, message='yyjPuke yyjPuke
```

```
yyjPuke yyjPuke'),
 Row(label=1.0, prediction=0.0, columnindex=885, message='yyjPuke'),
 Row(label=1.0, prediction=0.0, columnindex=886, message='it smell like
gasalinne to me'),
 Row(label=1.0, prediction=0.0, columnindex=904, message='need a hairdresser
following you'),
 Row(label=1.0, prediction=0.0, columnindex=910, message='HOW TO MAKE MONEY IS
THE ANSWER XD'),
 Row(label=1.0, prediction=0.0, columnindex=930, message='went from a 2 to a
5'),
 Row(label=1.0, prediction=0.0, columnindex=998, message='the grudge'),
 Row(label=1.0, prediction=0.0, columnindex=1004, message='WHATS THE BALD
PATCH'),
 Row(label=1.0, prediction=0.0, columnindex=1006, message='Ghost'),
 Row(label=1.0, prediction=0.0, columnindex=1014, message='that girl must be
like BRUHHH KEKW'),...
```

[51]:
```
#label = 0, prediction = 1
preds_labels_message.filter((preds_labels_message.label==0.0) &␣
 ↪(preds_labels_message.prediction==1.0)).collect()
```

[51]:
```
[Row(label=0.0, prediction=1.0, columnindex=133, message='Kappa'),
 Row(label=0.0, prediction=1.0, columnindex=134, message='Kappa'),
 Row(label=0.0, prediction=1.0, columnindex=135, message='!res'),
 Row(label=0.0, prediction=1.0, columnindex=136, message='!crosshair'),
 Row(label=0.0, prediction=1.0, columnindex=139, message='!crosshair'),
 Row(label=0.0, prediction=1.0, columnindex=143, message='haahaha'),
 Row(label=0.0, prediction=1.0, columnindex=144, message='so baddddddddd'),
 Row(label=0.0, prediction=1.0, columnindex=148, message='PRESS TAB'),
 Row(label=0.0, prediction=1.0, columnindex=149, message='Darkay>'),
 Row(label=0.0, prediction=1.0, columnindex=151, message='carried so
haaaaaard'),
 Row(label=0.0, prediction=1.0, columnindex=152, message='TAB'),
 Row(label=0.0, prediction=1.0, columnindex=153, message='basic white boy ego'),
 Row(label=0.0, prediction=1.0, columnindex=157, message='lfaooo'),
 Row(label=0.0, prediction=1.0, columnindex=158, message='10 seconds. he died
first'),
 Row(label=0.0, prediction=1.0, columnindex=159, message='ahahahah'),
 Row(label=0.0, prediction=1.0, columnindex=160, message='LUL LUL'),
 Row(label=0.0, prediction=1.0, columnindex=161, message='they calle it'),
 Row(label=0.0, prediction=1.0, columnindex=162, message='ASS'),
 Row(label=0.0, prediction=1.0, columnindex=167, message='ROD'),
 Row(label=0.0, prediction=1.0, columnindex=168, message='!ID'),
 Row(label=0.0, prediction=1.0, columnindex=169, message='can you stop crying
now? jesus'),
 Row(label=0.0, prediction=1.0, columnindex=170, message='@EditsByHunter you
```

```
forgot the L at the start.'),
 Row(label=0.0, prediction=1.0, columnindex=171, message='!mouse'),
 Row(label=0.0, prediction=1.0, columnindex=172, message='@EditsByHunter wrong
person D:'),
 Row(label=0.0, prediction=1.0, columnindex=175, message='mwahahahaha'),
 Row(label=0.0, prediction=1.0, columnindex=177, message='carried'),
 Row(label=0.0, prediction=1.0, columnindex=178, message='yoru carry, good
duelist'),
 Row(label=0.0, prediction=1.0, columnindex=179, message='#darkaynationn'),
 Row(label=0.0, prediction=1.0, columnindex=182, message='carried as always'),
 Row(label=0.0, prediction=1.0, columnindex=186, message='marcel424Razorlurk'),
 Row(label=0.0, prediction=1.0, columnindex=187, message='iitzDead'),
 Row(label=0.0, prediction=1.0, columnindex=188, message='14-26 KEKW'),
 Row(label=0.0, prediction=1.0, columnindex=190, message='1v11'),
 Row(label=0.0, prediction=1.0, columnindex=192, message='time to loose LUL'),
 Row(label=0.0, prediction=1.0, columnindex=193, message='iitzPepegaAim'),
 Row(label=0.0, prediction=1.0, columnindex=194, message='yea x is h on
cyrillic'),
 Row(label=0.0, prediction=1.0, columnindex=196, message='lmfao'),
 Row(label=0.0, prediction=1.0, columnindex=199, message='xddd'),
 Row(label=0.0, prediction=1.0, columnindex=200, message='real'),
 Row(label=0.0, prediction=1.0, columnindex=201, message='im immo btw'),
 Row(label=0.0, prediction=1.0, columnindex=202, message='LUL LUL'),
 Row(label=0.0, prediction=1.0, columnindex=203, message='HAHAHAA'),
 Row(label=0.0, prediction=1.0, columnindex=205, message='died every round
KEKW'),
 Row(label=0.0, prediction=1.0, columnindex=207, message='LUL LUL LUL LUL LUL
LUL  TRUE'),
 Row(label=0.0, prediction=1.0, columnindex=210, message='Weirdchamp'),
 Row(label=0.0, prediction=1.0, columnindex=310, message='HAHAHAHA'),
 Row(label=0.0, prediction=1.0, columnindex=311, message='LMFAOAOOAA'),
 Row(label=0.0, prediction=1.0, columnindex=312, message='KEKW'),
 Row(label=0.0, prediction=1.0, columnindex=314, message='palattt'),
 Row(label=0.0, prediction=1.0, columnindex=315, message='Shush'),
 Row(label=0.0, prediction=1.0, columnindex=316, message='Lmaooooooooooo'),
 Row(label=0.0, prediction=1.0, columnindex=317, message='died every round'),
 Row(label=0.0, prediction=1.0, columnindex=318, message='cryy rod'),
 Row(label=0.0, prediction=1.0, columnindex=319, message='cry'),
 Row(label=0.0, prediction=1.0, columnindex=321, message='CRY ABOUT IT BITCH'),
 Row(label=0.0, prediction=1.0, columnindex=324, message='maybe boring but
atleast im not jett spammer with 14-26 score'),
 Row(label=0.0, prediction=1.0, columnindex=326, message='done'),
 Row(label=0.0, prediction=1.0, columnindex=328, message='@ex1stinho imagine
saying proud im d2'),
 Row(label=0.0, prediction=1.0, columnindex=330, message="whoever's saying dog
ur moms doh"),
 Row(label=0.0, prediction=1.0, columnindex=332, message='LMAOOO YOU ARE SO
```

```
DOGWATER THATS WHY YOU GET TRIGGERED! I GET IT YOU GOT BULLIED IN SCHOOL BUT ITS
OK @prod'),
 Row(label=0.0, prediction=1.0, columnindex=333, message='Lowkey fax I'm
bronze'),
 Row(label=0.0, prediction=1.0, columnindex=334, message='i love you'),
 Row(label=0.0, prediction=1.0, columnindex=340, message='DUDE JUST SAID HE WONT
GET OFFENDED LIAR NotLikeThis'),
 Row(label=0.0, prediction=1.0, columnindex=341, message='!stats'),
 Row(label=0.0, prediction=1.0, columnindex=342, message='aceuClown'),
 Row(label=0.0, prediction=1.0, columnindex=344, message='no cap'),
 Row(label=0.0, prediction=1.0, columnindex=345, message='FACTS'),
 Row(label=0.0, prediction=1.0, columnindex=347, message='truee'),
 Row(label=0.0, prediction=1.0, columnindex=349, message='cry'),
 Row(label=0.0, prediction=1.0, columnindex=350, message='!uptime'),
 Row(label=0.0, prediction=1.0, columnindex=355, message='NotLikeThis
NotLikeThis NotLikeThis NotLikeThis NotLikeThis NotLikeThis NotLikeThis')...
```

Reflection

We find a couple surprising things: the model was not able to pick up automated messages like the one saying "Enjoying the stream? Subscribe to support the channel! https://www.twitch.tv/subs/jinnytty." This might be due to the fact that stripping the punctuation resulted in concatenation of the strings and the model not having picked up "https-swwwtwitchtvsubsjinnytty" due to either not having counted it enough or a vector size of 300 not being enough. Also, a lot of false positives should not have been classified as such. Words like bronze, crosshair and died are very popular words in PROD's channel. A lot of times we feel like the model would have gotten it right if it were more robust to alternative spellings, e.g. linking "badddddddd" to "bad", "to loose" to "to lose" or "cryy" to "cry" would have greatly improved the performance.

Some of these reasons could suggest that TF-IDF might have been a better approach than word2vec, but we made this conscious choice as we would like to contextually link words together. For example, when people talk about traps on Jinnytty's channel they usually refer to tourist traps, while on PROD's channel it could be an ability from a playable character like a "Cypher trap". This requires contextual linking. This would not be possible in TF-IDF.

4 Assignment 4

4.1 Introduction

The goal of this assignment was to investigate the popular social short video platform, TikTok. TikTok has been under a lot scrutiny since the Russia invasion of Ukraine. Like all the social media platform, it was also flooded with clips, news and videos coming from people in Ukraine, journalists, but also influencers and fake news accounts.

We were interested in knowing whether we could detect communities with specific interests. We detect communities with the Louvain Modularity algorithm and try to identify the main topics on which they post.

First, we import the packages required for this assignment.

[5]:
```
# for connecting jupyter notebook with neo4j
from py2neo import Graph

# for processing data and NLP
import pandas as pd
import numpy as np
import re
import gensim
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from gensim import corpora, models
from nltk.stem.porter import *
np.random.seed(2018)
from gensim.models import LdaMulticore, TfidfModel, CoherenceModel
from gensim.corpora import Dictionary
from gensim.models.phrases import Phrases

import time # to know how long training took
import multiprocessing # to speed things up by parallelizing

import nltk
nltk.download('wordnet')
```

In order to call neo4j from the jupyter notebook, first start the required database and run the following with database name as the name parameter and database password as the password parameter. Also add the APOC and the Graph Data Science Library plugins to the database.

[6]:
```
graph = Graph("bolt://localhost:7687",password="neo4j4u",name="neo4j")
```

4.2 Page Rank Algorithm

The Page Rank algorithm measures the importance of each node within the graph, based on the number incoming relationships and the importance of the corresponding source nodes. The

underlying assumption roughly speaking is that a node is only as important as the nodes that link to it.

PageRank is introduced in the original Google paper as a function that solves the following equation:

$$PR(A) = (1-d) + d(\frac{PR(T_1)}{C(T_1)} + ... + \frac{PR(T_n)}{C(T_n)})$$

where, we assume that a page A has pages T1 to Tn which point to it, d is a damping factor which can be set between 0 (inclusive) and 1 (exclusive) and C(A) is defined as the number of links going out of page A.

d is usually set to 0.85.

This equation is used to iteratively update a candidate solution and arrive at an approximate solution to the same equation.

First, we set the weights of the recommends edge to 1.

[7]:
```
graph.run("match (n:user)-[r:recommends]->(m:user) set r.weight=1")
```

Then, we project the graph with user as nodes and recommends as the edge. We also load properties on the edges. Then we stream page rank algorithm to return the page rank of each node.

[8]:
```
graph.run("CALL gds.graph.
 ↪project('myGraph_1','user','recommends',{relationshipProperties: 'weight'})")
result = graph.run("CALL gds.pageRank.stream('myGraph_1') YIELD nodeId, score␣
 ↪RETURN gds.util.asNode(nodeId).nickname AS name, score ORDER BY score DESC,␣
 ↪name ASC")
```

To view the page rank for all the nodes, we convert the result to a dataframe.

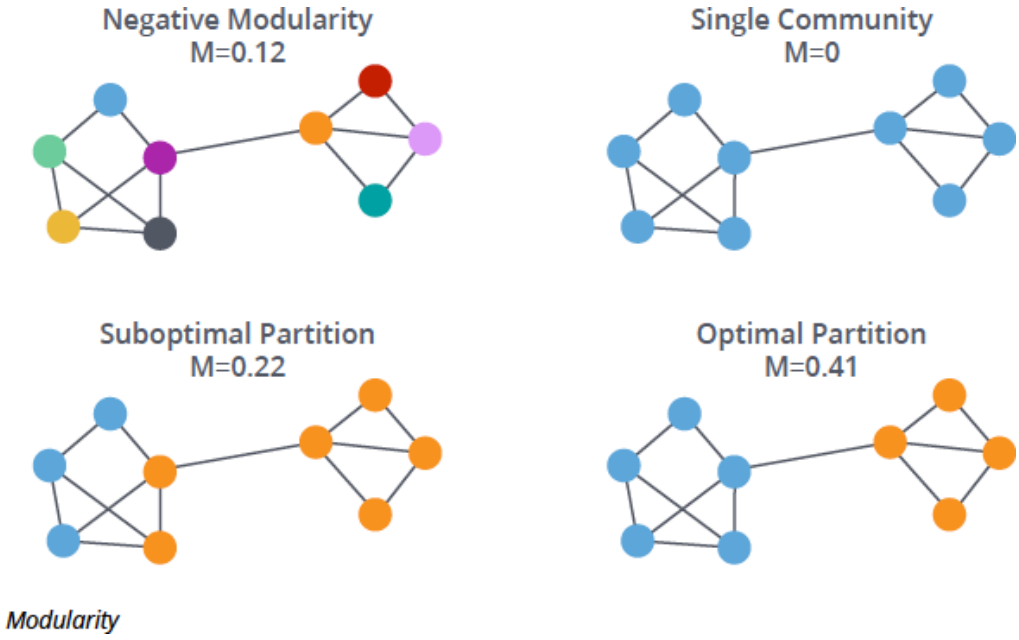
[9]:
```
page_ranks = result.to_data_frame()
page_ranks
```

[9]:
```
                   name     score
0       Malicia & Salva  0.924330
1       Nicolas Lacroix  0.917667
2           Celine Dept  0.912277
3                       0.896603
4                MAUDE  0.873413
...                ...       ...


[127224 rows x 2 columns]
```

Using page rank on users is inconclusive as the top users like Malicia & Salva, Nicholas Lacroix are all belgian and this is because of the fact that Tiktok is recommending based on the geolocation.

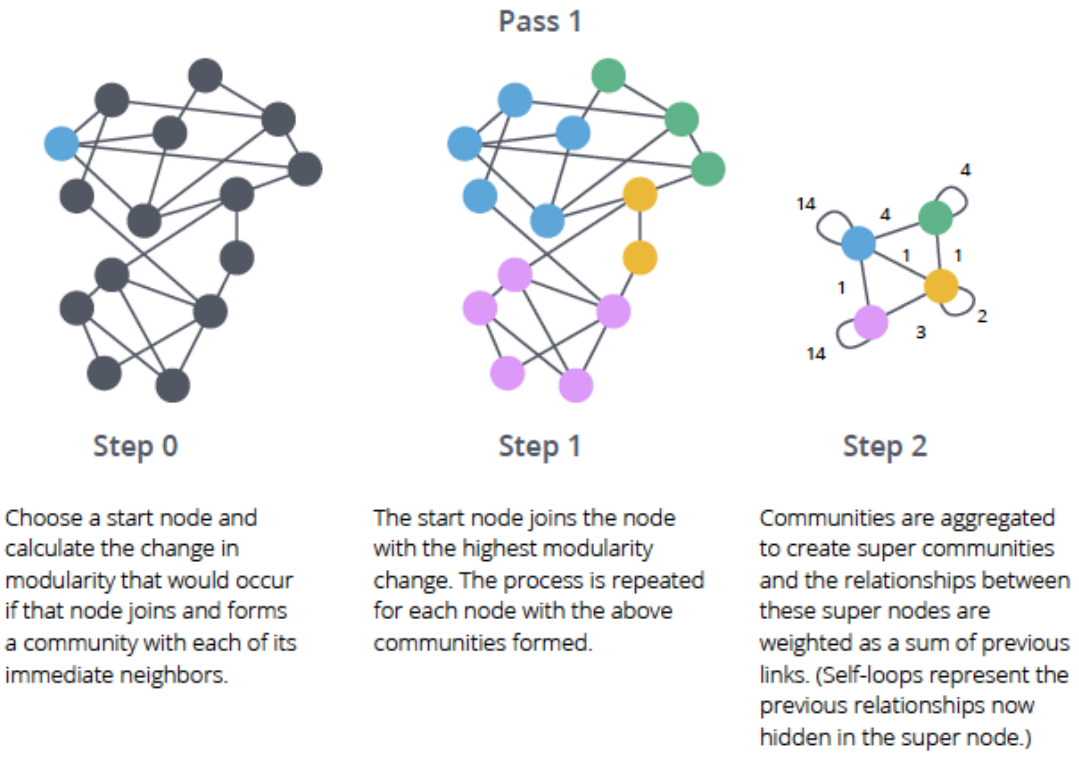
4.3 Community Detection

We try to find community of users who recommend one another using the Louvain Modularity algorithm. This algorithm maximizes a modularity score for each community, where the modularity quantifies the quality of an assignment of nodes to communities by evaluating how much more densely connected the nodes within a community are, compared to how connected they would be in a random network. The Louvain algorithm is one of the fastest modularity-based algorithms and works well with large graphs. It also reveals a hierarchy of communities at different scales, which is useful for understanding the global functioning of a network. In order to understand the Louvain Modularity algorithm, we look at what modularity is.



Modularity

Modularity is a measure of how well groups have been partitioned into clusters. It compares the relationships in a cluster compared to what would be expected for a random (or other baseline) number of connections.



Louvain Modularity Algorithm

4.3.1 Louvain Modularity

Now we create the projected graph and store it in the graph catalog. We load the recommends edge with orientation set to undirected as Louvain algorithm works best under this criteria.

[10]:
```
graph.run("CALL gds.graph.project('myGraph_2','user',{recommends: {orientation:␣
↪'UNDIRECTED'}},{relationshipProperties:'weight'})")
```

Now we stream the algorithm returning the community ID for each node. This allows us to inspect the results directly or post-process them in Cypher without any side effects.

[11]:
```
# To get each of the nodes and the community it belongs to uncomment the below␣
 ↪line
#result_1 = graph.run("CALL gds.louvain.stream('myGraph_1') YIELD nodeId,␣
 ↪communityId, intermediateCommunityIds RETURN gds.util.asNode(nodeId).id AS␣
 ↪name, communityId, intermediateCommunityIds ORDER BY name ASC")

# To get the members of the community and the community id of that community
result_2 = graph.run("CALL gds.louvain.stream('myGraph_2') YIELD nodeId,␣
 ↪communityId, intermediateCommunityIds RETURN collect(gds.util.asNode(nodeId).
 ↪id) as members, communityId, intermediateCommunityIds ORDER BY size(members)␣
 ↪DESC")
```

The write execution mode writes the community ID for each node as a property to the Neo4j database.

[12]:
```
graph.run("CALL gds.louvain.write('myGraph_2', { writeProperty: 'community' })␣
 ↪YIELD communityCount, modularity, modularities")
```

Though we get the stream mode returns to community ID for each node, we need to pass it as a dataframe to further investigate the communities.

[13]:
```
communities = result_2.to_data_frame()
communities
```

[13]:

	members	communityId	intermediateCommunityIds
0	[polakniko, dylanmto7, milxxtr, frankorando, d...	16843	None
1	[iwras, jeenie.weenie, dailymail, pandaboi.com...	99457	None
2	[dylanpage.ning, thedailyshow, brutofficiel, l...	17174	None
3	[thetiktokdrummer, idaarts, 01_funnyvids_, sem...	7671	None
4	[martavasyuta, skynews, fluro88, triggerpod, p...	103160	None
...	...	...	...
10804	[zelesnky88]	1399	None
10805	[zimnefrytki]	1400	None
10806	[thedogshouse]	1401	None
10807	[shmeebit]	1402	None
10808	[britishpromise.cats]	1421	None

```
10808                       None

[10809 rows x 3 columns]
```

The first community is a random community of tiktokers. They may or may not have posted on the war.

[17]:
```
communities[(communities["communityId"]==99457)]['members'][1][1:10]
```

[17]:
```
['jeenie.weenie',
 'dailymail',
 'pandaboi.com',
 'mamalindy',
 'pawsomepets',
 'visit',
 'anthonyriveras',
 'ninaishou',
 'timothytancre']
```

Next we have a community of news channels.

[18]:
```
communities[(communities["communityId"]==17174)]['members'][2][1:10]
```

[18]:
```
['thedailyshow',
 'brutofficiel',
 'lisaremillard',
 'quicktakenews',
 'goldbacked',
 'cbsmornings',
 'goodmorningbadnews',
 'taznii',
 'evankail']
```

And finally we see that we also have a community of military channels which posts military videos.

[20]:
```
communities[(communities["communityId"]==16843)]['members'][0][1:10]
```

[20]:
```
['dylanmto7',
 'milxxtr',
 'frankorando',
 'dimitriushakov',
 'fundiambb1',
 'iexcxity',
 'saih.one',
 'thatonegoogleearthdude',
 'leonsloonyeditz_']
```

[ ]:
```
#communities.to_csv('members.csv',header=True,index=False)
```

[21]:
```
#communities = pd.read_csv('members.csv')
```

[22]:
```
communities.drop(['intermediateCommunityIds'],axis=1,inplace=True)
```

[23]:
```
def tags_dict(communityid):
    """
    Function to get the dictionary of tags and the counts of the tags in the␣
 ↪posts of the users of the community
    """
    # retrieveing the members of the community
    txt = str(communities[communities['communityId']==communityid].values)
    x = re.findall(('[a-zA-Z0-9_\-\.]+'), txt)

    # Finding all the tags used by the members of this community
    d=pd.DataFrame()
    for name in x:
        result = graph.run("MATCH (n:user {id:$name})-->(:video)-->(t:tag)␣
 ↪RETURN t.title ",name=name)
        df = result.to_data_frame()
        d = pd.concat([d,df])

    # making a dictionary of the tags used by the members of the community.
    values = d['t.title'].value_counts(dropna=False).keys().tolist()
    counts = d['t.title'].value_counts(dropna=False).tolist()
    value_dict = dict(zip(values, counts))

    return value_dict,d
```

Now let us explore the tags used by members of the top three communities.

[27]:
```
value_dict_0, d_0 = tags_dict(99457)
value_dict_0
```

[27]:
```
{'fyp': 914,
 'foryou': 653,
 'viral': 350,
 'foryoupage': 350,
 'funny': 136,
 'comedy': 130,
 'fyp': 128,
 'satisfying': 120,
 'trending': 109,
 'asmr': 107,
 'tiktok': 105,
 'fy': 75,
 'stitch': 67,
 'xyzbca': 67,
 'duet': 63,
```

```
 '4u': 62,
 'ukraine': 57,
 ...}
```

The dictionary above shows that the most used tags by the tiktokers are fyp, foryou, foryoupage, viral and so on. This is well in line with their objective to reach as many tiktok users as possible. Using the tag #fyp or other forms of it show the video that you post in other users recommendation which helps to increase their views.

[28]:
```
value_dict_1,d_1 = tags_dict(17174)
value_dict_1
```

[28]:
```
{'fyp': 526,
 'news': 343,
 'ukraine': 275,
 'russia': 229,
 'ukrainewar': 118,
 'foryou': 101,
 'putin': 101,
 'politics': 101,
 'greenscreen': 94,
 'biden': 82,
 'war': 80,
 'foryoupage': 71,
 'stitch': 63,
 'fyp': 59,
 'ukraine': 58,
 'science': 56,
 ...}
```

The output above shows that the most used tags by the community of news channels are fyp, news, ukraine, russia, politics, money science and so on. This is also in line with our expectations.

[29]:
```
value_dict_2, d_2 =tags_dict(16843)
value_dict_2
```

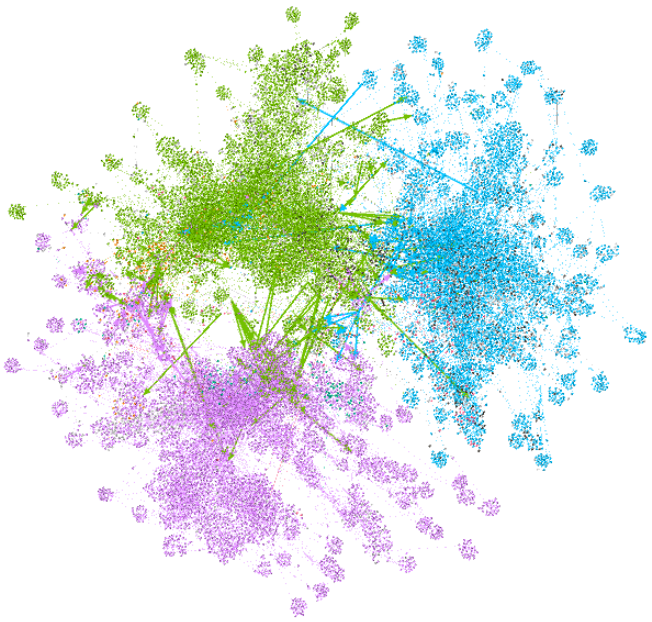
[29]:
```
{'fyp': 839,
 'ukraine': 322,
 'foryou': 306,
 'military': 289,
 'viral': 288,
 'russia': 277,
 'foryoupage': 255,
 'fyp': 229,
 'war': 160,
 'edit': 155,
 'ww3': 125,
 'history': 124,
```

```
'nato': 122,
'army': 114,
'xyzbca': 113,
'usa': 110,
...}
```

The output above is also in line with the expectations. Though there are tags like fyp, foryoupage and so on, these can be seen as tags to increase the reach of the posts. The main topics in which they post are ukrain, military, viral, russia, war, edit, ww3 and so on.

4.3.2 Visualization

The three main communities that we considered look as below. We see that the communities are dense, with very few connections between them. We can also see that inside each of the main communities we can see the smaller modules.



4.4 Topic Modeling using Latent Dirichelet Allocation (LDA)

As we saw above, the topic each community posts on can be identified by looking at the tags used by the members of the community. To make it easier to identify the topics the community posts in, we try using Latent Dirichelet Allocation. To begin, we first load all the descriptions of all the videos.

[30]:
```
tags = graph.run("MATCH (t:video) RETURN t.desc")
tags_df = tags.to_data_frame().set_index('t.desc').reset_index()
```

Now we preprocess the descriptions. We split the sentences to words and then retrieve the stem form of the words. We also remove the stopwords from this.

[31]:
```
stemmer = SnowballStemmer('english')

def lemmatize_stemming(text):
    return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))

def preprocess(text):
    result = []
    for token in gensim.utils.simple_preprocess(text):
        if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) >␣
 ↪3:
            result.append(lemmatize_stemming(token))
    return result
```

Let us look at an example.

[33]:
```
desc_sample = tags_df.values[0][0]
print('original document: ')
words = []
for word in desc_sample.split(' '):
    words.append(word)
print(words)
print('\n\n tokenized and lemmatized document: ')
print(preprocess(desc_sample))
```

```
original document:
['', '#Ukraine', '', 'Spread', 'awareness!']


 tokenized and lemmatized document:
['ukrain', 'spread', 'awar']
```

Now we apply preprocessing to all the descriptions.

[34]:
```
processed_desc = tags_df['t.desc'].map(preprocess)
processed_desc[1:10]
```

[34]:
```
1                                    [russia, ukrain]
2    [live, best, life, thank, russia, ukrain, stop...
3    [russia, ukrain, ongo, seri, russia, ukrain, n...
4    [parent, havnt, come, sister, cri, scar, ukrai...
5              [spread, awar, support, ukrain, ukrain]
6              [russia, ukrain, russiavsukrain, viral]
7                                    [ukrain, georgia]
8                  [tear, come, lose, ukrain, russia]
```

```
9                                        [ukrain, elbruso]
Name: t.desc, dtype: object
```

Then we create a dictionary from the processed descriptions.

[35]:
```
dictionary = gensim.corpora.Dictionary(processed_desc)
```

[36]:
```
count = 0
for k, v in dictionary.iteritems():
    print(k, v)
    count += 1
    if count > 10:
        break
```

```
0 awar
1 spread
2 ukrain
3 russia
4 best
5 life
6 live
7 russiastop
8 stopwar
9 thank
10 nato
```

Next we filter the dictionary to keep words that are not too frequent or not too rare.

[37]:
```
dictionary.filter_extremes(no_below=15, no_above=0.5, keep_n=100000)
```

Now we convert the processed descriptions to numerical ID's and their frequency.

[38]:
```
bow_corpus = [dictionary.doc2bow(doc) for doc in processed_desc]
bow_corpus[4310]
```

[38]:
```
[(93, 1), (807, 1), (4204, 1)]
```

[39]:
```
bow_doc_4310 = bow_corpus[4310]

for i in range(len(bow_doc_4310)):
    print("Word {} (\"{}\") appears {} time.".format(bow_doc_4310[i][0],
                                                     ␣
 ↪dictionary[bow_doc_4310[i][0]],
                                                      bow_doc_4310[i][1]))
```

```
Word 93 ("need") appears 1 time.
Word 807 ("friend") appears 1 time.
Word 4204 ("answer") appears 1 time.
```

Next we create a TF-IDF model using the generated bag of words corpus and convert the bag of words corpus to a tfidf corpus.

[40]:
```
tfidf = models.TfidfModel(bow_corpus)
corpus_tfidf = tfidf[bow_corpus]
```

Let us look how the first description look under different corpuses.

[41]:
```
# visualization of different preprocessing steps
print(processed_desc[0]) # original input
print()
print(bow_corpus[0]) # hashed BOW version
print()
print(corpus_tfidf[0]) # TFIDF-weighted version
```

```
['ukrain', 'spread', 'awar']

[(0, 1), (1, 1), (2, 1)]

[(0, 0.6533663825109627), (1, 0.6970005567953172), (2, 0.29547012375459564)]
```

[42]:
```
# coherence_values = []

# dev_size = 15000

# for num_topics in range(5, 16):
#     model = LdaMulticore(corpus=corpus_tfidf[:dev_size],
#                          id2word=dictionary,
#                          num_topics=num_topics, random_state=42)

#     coherencemodel_umass = CoherenceModel(model=model,
#                                           texts=processed_desc[:dev_size],
#                                           dictionary=dictionary,
#                                           coherence='u_mass')

#     coherencemodel_cv = CoherenceModel(model=model,
#                                        texts=processed_desc[:dev_size],
#                                        dictionary=dictionary,
#                                        coherence='c_v')

#     umass_score = coherencemodel_umass.get_coherence()
#     cv_score = coherencemodel_cv.get_coherence()

#     print(num_topics, umass_score, cv_score)
#     coherence_values.append((num_topics, umass_score, cv_score))
```

[43]:
```
# %matplotlib inline
# import matplotlib.pyplot as plt
# import seaborn
```

```
# seaborn.set_context('poster') # use large font


# scores = pd.DataFrame(coherence_values, columns=['num_topics', 'UMass', 'CV'])
# # scores['UMass'] = -scores['UMass']
# fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 10))
# scores.plot.line(x='num_topics', y='UMass', ax=ax[0], xticks=range(5,16));
# scores.plot.line(x='num_topics', y='CV', ax=ax[1], xticks=range(5,16));
```

Now we create the lda model using the bag of words corpus.

[74]:
```
lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=7,␣
 ↪id2word=dictionary, passes=2, workers=2)
for idx, topic in lda_model.print_topics(-1):
    print('Topic: {} \nWords: {}'.format(idx, topic))
```

```
Topic: 0
Words: 0.063*"pourtoi" + 0.022*"répondr" + 0.018*"humour" + 0.016*"vous" +
0.015*"pour" + 0.011*"avec" + 0.010*"foryou" + 0.008*"insta" + 0.008*"dan" +
0.007*"parti"
Topic: 1
Words: 0.069*"ukrain" + 0.036*"russia" + 0.027*"repli" + 0.015*"ukrainewar" +
0.014*"news" + 0.012*"putin" + 0.010*"russian" + 0.009*"" +
0.009*"militari" + 0.008*"foryou"
Topic: 2
Words: 0.015*"game" + 0.015*"" + 0.011*"happi" + 0.010*"prank" +
0.009*"year" + 0.009*"stitch" + 0.008*"easter" + 0.007*"foryou" + 0.007*"like" +
0.006*"fail"
Topic: 3
Words: 0.023*"asmr" + 0.021*"satisfi" + 0.012*"repli" + 0.009*"work" +
0.009*"time" + 0.009*"good" + 0.008*"foryou" + 0.007*"clean" + 0.007*"life" +
0.007*"come"
Topic: 4
Words: 0.089*"foryou" + 0.073*"viral" + 0.069*"foryoupag" + 0.059*"fyp" +
0.030*"funni" + 0.029*"trend" + 0.019*"anim" + 0.017*"xyzbca" + 0.015*"comedi" +
0.013*"tiktok"
Topic: 5
Words: 0.039*"foryou" + 0.032*"fyp" + 0.025*"viral" + 0.021*"duet" +
0.018*"movi" + 0.016*"tiktok" + 0.016*"food" + 0.014*"foryoupag" +
0.013*"capcut" + 0.012*"film"
Topic: 6
Words: 0.035*"foryou" + 0.027*"viral" + 0.023*"fyp" + 0.018*"foryoupag" +
0.010*"trend" + 0.010*"tiktok" + 0.010*"repli" + 0.009*"love" + 0.007*"want" +
0.006*"know"
```

4.4.1 Community Identification

Now to see how good the model performs, we look at its performance on a sample description

[89]:
```
s=processed_desc[10]
t=dictionary.doc2bow(s)
```

[91]:
```
for index, score in sorted(lda_model[t], key=lambda tup: -1*tup[1]):
    print("\nScore: {}\t \nTopic: {}".format(score, lda_model.print_topic(index,␣
 ↪10)))
```

```
Score: 0.5154715776443481
Topic: 0.069*"ukrain" + 0.036*"russia" + 0.027*"repli" + 0.015*"ukrainewar" +
0.014*"news" + 0.012*"putin" + 0.010*"russian" + 0.009*"" +
0.009*"militari" + 0.008*"foryou"

Score: 0.23831318318843842
Topic: 0.035*"foryou" + 0.027*"viral" + 0.023*"fyp" + 0.018*"foryoupag" +
0.010*"trend" + 0.010*"tiktok" + 0.010*"repli" + 0.009*"love" + 0.007*"want" +
0.006*"know"

Score: 0.11507231742143631
Topic: 0.015*"game" + 0.015*"" + 0.011*"happi" + 0.010*"prank" +
0.009*"year" + 0.009*"stitch" + 0.008*"easter" + 0.007*"foryou" + 0.007*"like" +
0.006*"fail"

Score: 0.09519561380147934
Topic: 0.063*"pourtoi" + 0.022*"répondr" + 0.018*"humour" + 0.016*"vous" +
0.015*"pour" + 0.011*"avec" + 0.010*"foryou" + 0.008*"insta" + 0.008*"dan" +
0.007*"parti"
...
```

We see that the model works well when using single tags. Now let us see when we use the tags by each community.

[131]:
```
s= d_2['t.title'].tolist()
t=dictionary.doc2bow(s)
```

The topic with the highest score, which appears at the top, is the topic suggested by the model.

[132]:
```
for index, score in sorted(lda_model[t], key=lambda tup: -1*tup[1]):
    print("\nScore: {}\t \nTopic: {}".format(score, lda_model.print_topic(index,␣
 ↪10)))
```

```
Score: 0.35648566484451294
Topic: 0.069*"ukrain" + 0.036*"russia" + 0.027*"repli" + 0.015*"ukrainewar" +
0.014*"news" + 0.012*"putin" + 0.010*"russian" + 0.009*"" +
0.009*"militari" + 0.008*"foryou"

Score: 0.3099782466888428
Topic: 0.089*"foryou" + 0.073*"viral" + 0.069*"foryoupag" + 0.059*"fyp" +
```

```
0.030*"funni" + 0.029*"trend" + 0.019*"anim" + 0.017*"xyzbca" + 0.015*"comedi" +
0.013*"tiktok"

Score: 0.12473056465387344
Topic: 0.039*"foryou" + 0.032*"fyp" + 0.025*"viral" + 0.021*"duet" +
0.018*"movi" + 0.016*"tiktok" + 0.016*"food" + 0.014*"foryoupag" +
0.013*"capcut" + 0.012*"film"

Score: 0.08104458451271057
Topic: 0.015*"game" + 0.015*"" + 0.011*"happi" + 0.010*"prank" +
0.009*"year" + 0.009*"stitch" + 0.008*"easter" + 0.007*"foryou" + 0.007*"like" +
0.006*"fail"
...
```

For the first community of tiktokers, we see that the model identifies the topic almost correctly with a score of 0.374. Though this is not above 0.5, the model is able to correctly identify that the members of this community mainly post in topic at the top which consists of the viral tags. The score of the first community is not above 0.5 as the members of any community posts with tags in all the categories. So all the communities could be expected to have a score of below 0.5

For the second community of news channels, we see that the model identifies the topic almost correctly with a score of 0.43. Again the model is able to correctly identify that the members of this community mainly post in topic at the top which consists of the mainly the russian invasion tags and the news tag.

For the third community of military channels, we see that the model identifies the topic almost correctly with a score of 0.356. Again the model is able to correctly identify that the members of this community mainly post in topic at the top which consists of the mainly the russian invasion tags and the news tag. But the score of the second topic is also above 0.30 suggesting that the members of this community posts in the popular fyp tags as well to make their posts popular.

4.5 References

- Graph Databases
- A blog on LDA
- A blog on Louvain Modularity
- Community Detection Algorithms
- Page Rank Algorithm