Project: Radio link failure prediction

By *P R O D A L*

Yassine Hamdaoui-Eya Amri-Marouene-Guezmir-Bilel Mensi-Khalil Allah Abbes

2021-2022

Table des matières

Introduction

The goal of our project is to limit the radio link failure by exploiting the data provided by Turkcell to guarantee an ideal environment.

To ensure an ideal transmission environment we will implement several models to predict radio link failures.

Business Objectives

- Ensure performance of the network by avoiding network failures
- Reduce the effect of the weather on the performance of the radio link
- Financial gain and customer satisfaction

Data science Objectives

- Predict radio link failure for next day and next 5 days based on region, historical data and weather data.
- Implement a prediction system to identify the causes of the radio link failure
- Identify weather conditions that affect directly the network
- Anticipate the settings to be made on the radio stations to avoid radio link failure

Presentation of the company

Turkcell is the leading mobile phone operator of Turkey, based in Istanbul. The company has 39,3 million subscribers as of September 2021. In 2015, the company's number of subscribers climbed to 68.9 million, in nine countries largest shareholder is Turkey Wealth Fund with 26.2% ownership. It is one of the world's biggest companies (Fortune 2000) list published by Fortune Turkcell has also developed Yaani, a browser for mobile and desktop. Turkcell's general manager is Murat Erkan.

Background

- As 5G mobile networks are getting to be spread globally, the stable and high-quality operation is necessary to minimize the 5G service failure.

- In that situation, network automation is a key to accelerate 5G network penetration, although highly experienced operators can tackle affected network failure and the anomaly detection is additionally desired to be automatically and rapidly performed by AI/ML.

Radio Link failure

Radio Link Failure (RLF) is a challenging problem in 5G networks as it may decrease communication reliability and increases latency. This is against the objectives of 5G, particularly for the ultra-Reliable Low Latency Communications (uRLLC) traffic class. RLF can be predicted using radio measurements reported by User Equipment (UE)s, such as Reference Signal Receive Power (RSRP), Reference Signal Receive Quality (RSRQ), Channel Quality Indicator (CQI), and Power HeadRoom (PHR).

UE may assume that Radio Link is broken in the following setuation.

- The measured RSRP is too low (under a certain limit)
- It failed to decode PDCCH due to power signal quality (e.g, low RSRP, RSRQ)
- It failed to decode PDSCH due to power signal quality (e.g, low RSRP, RSRQ)

However, detailed mechanism to RLF is up to the chipset implementation. So the individual RLF detection mechanism may vary chipset-to-chipset.

- eNodeB may assume that that Radio Link is broken in the following setuation.
- SRS Power (SINR) from UE is much lower than what eNB configured for the UE
- eNodeB couldn't detect (see) any NACK nor ACK from UE for PDSCH.

Data preparation

Loading the original data

```
%%time
# Around 1mins
xlsx = pd.ExcelFile('data-Test-2020-09-01.xlsx')
sheets = pd.read_excel(xlsx, sheet_name=None, index_col=0,
                       na_filter=True, convert_float=False)

met_stations = sheets['met-stations']
met_real = sheets['met-real']
met_forecast = sheets['met-forecast']
rl_sites = sheets['rl-sites']
rl_kpis = sheets['rl-kpis']
distances = sheets['distances']

# Station names
stations = met_stations['station_no'].tolist()

# Forecast dates
forecast_dates = sorted(list(set(met_forecast['datetime'])))
```

On MET Forecast:

- One value per each (station_no, datetime)
- The value will be the mean of the values per each (station_no, datetime)
- One hot encoding for the categorical data
- Only the forecast of the next day would be consider

Numerical data:

In [ ]:
```
met_forecast.sort_values(by=['station_no', 'datetime']).head()
```

Out[ ]:

	station_no	datetime	report_time	weather_day1	temp_max_day1	temp_min_day1	humidity_max_day1	humidity_min_day1	wind_dir_day1	wind_speed_day1	weather_day2
0.0	WS_17038	2020-01-02	evening	rain	10.0	7.0	NaN	NaN	NaN	NaN	overcast clouds
1.0	WS_17038	2020-01-02	morning	rain	10.0	7.0	NaN	NaN	NaN	NaN	overcast clouds
96.0	WS_17038	2020-01-03	evening	rain	10.0	7.0	78.0	68.0	322.0	11.0	scattered clouds
97.0	WS_17038	2020-01-03	morning	overcast clouds	10.0	7.0	78.0	68.0	322.0	11.0	scattered clouds
192.0	WS_17038	2020-01-04	evening	scattered clouds	10.0	7.0	85.0	65.0	41.0	11.0	scattered clouds

In [ ]:
```
to_drop = [c for c in met_forecast.columns
           if ('day1' not in c) and (c not in ['station_no', 'datetime', 'report_time'])]
met_forecast_v2 = met_forecast.drop(columns=to_drop)
met_forecast_v2.head()
```

Out[ ]:

	station_no	datetime	report_time	weather_day1	temp_max_day1	temp_min_day1	humidity_max_day1	humidity_min_day1	wind_dir_day1	wind_speed_day1
0.0	WS_17038	2020-01-02	evening	rain	10.0	7.0	NaN	NaN	NaN	NaN
1.0	WS_17038	2020-01-02	morning	rain	10.0	7.0	NaN	NaN	NaN	NaN
2.0	WS_17232	2020-01-02	evening	scattered clouds	12.0	5.0	NaN	NaN	NaN	NaN
3.0	WS_17232	2020-01-02	morning	scattered clouds	12.0	5.0	NaN	NaN	NaN	NaN
4.0	WS_17233	2020-01-02	evening	scattered clouds	13.0	6.0	NaN	NaN	NaN	NaN

In [ ]:
```
mean_values = met_forecast_v2.groupby(by=['station_no', 'datetime']).mean().reset_index()
mean_values.sort_values(by=['station_no', 'datetime']).head()
```

Out[ ]:

	station_no	datetime	temp_max_day1	temp_min_day1	humidity_max_day1	humidity_min_day1	wind_dir_day1	wind_speed_day1
0	WS_17038	2020-01-02	10.0	7.0	NaN	NaN	NaN	NaN
1	WS_17038	2020-01-03	10.0	7.0	78.0	68.0	322.0	11.0
2	WS_17038	2020-01-04	10.0	7.0	85.0	65.0	41.0	11.0
3	WS_17038	2020-01-05	12.0	6.0	84.0	59.0	155.0	10.0
4	WS_17038	2020-01-06	13.0	9.0	60.0	50.0	161.0	16.0

Categorical data

In [ ]:
```
numerical_values = met_forecast_v2.describe().columns
non_num_df = met_forecast_v2[[x for x in met_forecast_v2.columns
                              if x not in numerical_values]]
non_num_df.head()
```

Out[ ]:

	station_no	datetime	report_time	weather_day1
0.0	WS_17038	2020-01-02	evening	rain
1.0	WS_17038	2020-01-02	morning	rain
2.0	WS_17232	2020-01-02	evening	scattered clouds
3.0	WS_17232	2020-01-02	morning	scattered clouds
4.0	WS_17233	2020-01-02	evening	scattered clouds

In [ ]:
```
# 'Report_time' will be dropped
ohe_df = copy.deepcopy(non_num_df[['station_no', 'datetime']])
for x in range(1):
  temp = pd.get_dummies(non_num_df[f'weather_day{x+1}'], prefix=f'wd{x+1}')
  ohe_df = pd.concat([ohe_df, temp], axis=1)
ohe_df.head()
```

Out[ ]:

	station_no	datetime	wd1_few clouds	wd1_heavy rain	wd1_heavy rain showers	wd1_heavy snow	wd1_heavy thunderstorm with rain showers	wd1_light intensity shower rain	wd1_light rain showers	wd1_light snow	wd1_overcast clouds	wd1_rain	wd1_scattered clouds	wd1_sleet	wd1_
0.0	WS_17038	2020-01-02	0	0	0	0	0	0	0	0	0	1	0	0	
1.0	WS_17038	2020-01-02	0	0	0	0	0	0	0	0	0	1	0	0	
2.0	WS_17232	2020-01-02	0	0	0	0	0	0	0	0	0	0	1	0	
3.0	WS_17232	2020-01-02	0	0	0	0	0	0	0	0	0	0	1	0	
4.0	WS_17233	2020-01-02	0	0	0	0	0	0	0	0	0	0	1	0	

In [ ]:
```
# Dealing with two datetimes
ohe_df = ohe_df.groupby(by=['station_no', 'datetime'], as_index=False).agg(lambda x: 1 if sum(x) > 1 else sum(x))
ohe_df.head()
```

Out[ ]:

	station_no	datetime	wd1_few clouds	wd1_heavy rain	wd1_heavy rain showers	wd1_heavy snow	wd1_heavy thunderstorm with rain showers	wd1_light intensity shower rain	wd1_light rain showers	wd1_light snow	wd1_overcast clouds	wd1_rain	wd1_scattered clouds	wd1_sleet	wd1_sn
0	WS_17038	2020-01-02	0	0	0	0	0	0	0	0	0	1	0	0	
1	WS_17038	2020-01-03	0	0	0	0	0	0	0	0	1	1	0	0	
2	WS_17038	2020-01-04	0	0	0	0	0	0	0	0	0	0	1	0	
3	WS_17038	2020-01-05	0	0	0	0	0	0	0	0	0	0	1	0	
4	WS_17038	2020-01-06	0	0	0	0	0	0	0	0	1	0	0	0	

Combining the numerical data with the categorical:

In [ ]:
```
modified_forecast_df = pd.merge(left=mean_values, right=ohe_df, on=['station_no', 'datetime'])
modified_forecast_df.head()
```

Out[ ]:

	station_no	datetime	temp_max_day1	temp_min_day1	humidity_max_day1	humidity_min_day1	wind_dir_day1	wind_speed_day1	wd1_few clouds	wd1_heavy rain	wd1_heavy rain showers	wd1_heavy snow
0	WS_17038	2020-01-02	10.0	7.0	NaN	NaN	NaN	NaN	0	0	0	0
1	WS_17038	2020-01-03	10.0	7.0	78.0	68.0	322.0	11.0	0	0	0	0
2	WS_17038	2020-01-04	10.0	7.0	85.0	65.0	41.0	11.0	0	0	0	0
3	WS_17038	2020-01-05	12.0	6.0	84.0	59.0	155.0	10.0	0	0	0	0
4	WS_17038	2020-01-06	13.0	9.0	60.0	50.0	161.0	16.0	0	0	0	0

On RL KPIS

Using modified data provided by JSP
Adding the forecast information from the nearest weather stations (from the previous day)
Adding the KPI information from the previous day. (added by JSP)

In [ ]:
```
# To find nearest station
def find_nearest_stations(site_id: str, distances: pd.DataFrame,
                          stations: list, k: int = 1) -> str:
  temp = distances[[site_id]].sort_values(by=[site_id])
  temp = temp.loc[[x for x in temp.index if x in stations]].head(k)
  return list(temp.index)
```

In [ ]:
```
%%time
# Around 1mins
xlsx = pd.ExcelFile('data-Test-2020-09-01.xlsx')
rl_kpis_mod = pd.read_excel(xlsx, sheet_name='rl-kpis', index_col=0,
                        na_filter=True, convert_float=False)
rl_sites = pd.read_excel(xlsx, sheet_name='rl-sites', index_col=0,
                        na_filter=True, convert_float=False)
```

```
CPU times: user 25.3 s, sys: 154 ms, total: 25.5 s
Wall time: 25.5 s
```

In [ ]:
```
# Some columns were dropped
rl_kpis_mod = rl_kpis_mod.drop(['direction', 'neid'], axis = 1)
rl_kpis_mod.head()
```

Out[ ]:

	type	datetime	tip	mlid	mw_connection_no	site_id	polarization	card_type	adaptive_modulation	freq_band	link_length	severaly_error_second	error_second	unavail_
0.0	ENK	2020-01-02	NEAR	R0AP	314493.0	RL_9;SIP	Vertical	cardtype4	Enable	f3	2820.0	0.0	0.0	
1.0	ENK	2020-01-02	FAR	R1AG	227760.0	RL_I7LB>	Vertical	cardtype4	Enable	f3	5898.0	0.0	0.0	
2.0	ENK	2020-01-02	NEAR	R1BA	345715.0	RL_I7LB>	Vertical	cardtype4	Enable	f3	5134.0	0.0	0.0	
3.0	ENK	2020-01-02	NEAR	R0TR	265780.0	RL_I7T?Q	Vertical	cardtype4	Enable	f5	611.0	0.0	0.0	
4.0	ENK	2020-01-02	NEAR	R3XD	335068.0	RL_I7Z?H	Vertical	cardtype1	Enable	f4	1873.0	0.0	0.0	

In [ ]:
```
# Extend dates a day ahead.
rl_kpis_mod_addendum = rl_kpis_mod.loc[rl_kpis_mod['datetime'] == '2020-01-15',:].copy()
rl_kpis_mod_addendum['datetime'] = [x + pd.Timedelta(days=1) for x in rl_kpis_mod_addendum['datetime']]
rl_kpis_mod = pd.concat([rl_kpis_mod,rl_kpis_mod_addendum])
```

In [ ]:
```
%%time
# Around 2mins
# Forecast datetime should be - 1 day from the kpis datetime
rl_kpis_mod['forecast_datetime'] = [x - pd.Timedelta(days=1) for x in rl_kpis_mod['datetime']]

# Getting nearest station (just 1) based on the antennas - This might take a while
rl_kpis_mod['nearest_station'] = [find_nearest_stations(site_id, distances, stations)[0] for site_id in rl_kpis_mod['site_id']]

# Assuring the dates are timestamp type
modified_forecast_df['datetime'] = [pd.Timestamp(x) for x in modified_forecast_df['datetime']]
rl_kpis_mod['forecast_datetime'] = [pd.Timestamp(x) for x in rl_kpis_mod['forecast_datetime']]

# Modify name to merge
modified_forecast_df.rename(columns={'datetime':'forecast_datetime', 'station_no':'nearest_station'}, inplace=True)
```

```
CPU times: user 2min 24s, sys: 38.6 ms, total: 2min 24s
Wall time: 2min 24s
```

In [ ]:
```
## KPI Historical (one day) (Added by JSP)
rl_kpis_history = rl_kpis_mod.copy()

# Dropping columns.
rl_kpis_history.drop(columns = ['forecast_datetime','nearest_station'],inplace=True)

# Assuring the dates are timestamp type
rl_kpis_history['datetime'] = [pd.Timestamp(x) for x in rl_kpis_history['datetime']]

# Adding with site data.
rl_kpis_history = rl_kpis_history.merge(rl_sites[['site_id','groundheight','clutter_class']], on='site_id')

# Renaming columns for merging.
rl_kpis_history.columns= ['history_{}'.format(column) for column in rl_kpis_history.columns]

rl_kpis_history
```

Out[ ]:

	history_type	history_datetime	history_tip	history_mlid	history_mw_connection_no	history_site_id	history_polarization	history_card_type	history_adaptive_modulation	hist
0	ENK	2020-01-02	NEAR	R0AP	314493.0	RL_9;SIP	Vertical	cardtype4	Enable	
1	ENK	2020-01-03	NEAR	R0AP	314493.0	RL_9;SIP	Vertical	cardtype4	Enable	
2	ENK	2020-01-06	NEAR	R0AP	314493.0	RL_9;SIP	Vertical	cardtype4	Enable	
3	ENK	2020-01-07	NEAR	R0AP	314493.0	RL_9;SIP	Vertical	cardtype4	Enable	
4	ENK	2020-01-13	NEAR	R0AP	314493.0	RL_9;SIP	Vertical	cardtype4	Enable	
...	...	...	...	...	...	...	...	...	...	...
29445	ENK	2020-01-14	FAR	U3WB	1349170.0	RL_aENGH	Vertical	cardtype4	Enable	
29446	NEC	2020-01-15	NEAR	U3BT	339402.0	RL_aENGH	Vertical	cardtype5	Enable	
29447	ENK	2020-01-15	FAR	U3WB	1349170.0	RL_aENGH	Vertical	cardtype4	Enable	
29448	NEC	2020-01-16	NEAR	U3BT	339402.0	RL_aENGH	Vertical	cardtype5	Enable	
29449	ENK	2020-01-16	FAR	U3WB	1349170.0	RL_aENGH	Vertical	cardtype4	Enable	

29450 rows × 23 columns

In [ ]:

```
## Replacing Nearest Stations without Forecast

# weather station names.
ws = [i for i in distances.index if 'WS' in i]

# distances to weather stations with forecast.
ws_with_forecast = np.intersect1d(rl_kpis_mod['nearest_station'].unique(),modified_forecast_df['nearest_station'].unique())
distances_ws_with_forecast = distances.loc[ws,ws_with_forecast].copy()

# weather stations without forecast.
ws_without_forecast = np.setdiff1d(rl_kpis_mod['nearest_station'].unique(),modified_forecast_df['nearest_station'].unique())
ws_replacement = [(i,distances_ws_with_forecast.loc[i].index[distances_ws_with_forecast.loc[i].argmin()]) for i in ws_without_forecast]

# replacing in dataset.
for i in ws_replacement:
  rl_kpis_mod.loc[rl_kpis_mod['nearest_station'] == i[0],'nearest_station'] = i[1]
```

Merging the datasets

In [ ]:

```
### Some values seems to be dropped
merged_df = pd.merge(rl_kpis_mod, modified_forecast_df,
                     on=['nearest_station','forecast_datetime'],
                     validate='m:m')
merged_df
```

Out[ ]:

	type	datetime	tip	mlid	mw_connection_no	site_id	polarization	card_type	adaptive_modulation	freq_band	link_length	severaly_error_second	error_second	un
0	ENK	2020-01-03	NEAR	R0AP	314493.0	RL_9;SIP	Vertical	cardtype4	Enable	f3	2820.0	0.0	0.0	
1	ENK	2020-01-03	NEAR	R3XD	335068.0	RL_I7Z?H	Vertical	cardtype1	Enable	f4	1873.0	0.0	0.0	
2	ENK	2020-01-03	FAR	R0DK	306403.0	RL_I;YNL	Vertical	cardtype4	Enable	f4	1562.0	0.0	0.0	
3	ENK	2020-01-03	FAR	R0ED	313790.0	RL_I;YNL	Vertical	cardtype1	Enable	f3	3203.0	0.0	0.0	
4	ENK	2020-01-03	FAR	R0GM	1377555.0	RL_I;YNL	Vertical	cardtype1	Enable	f3	4866.0	0.0	0.0	

On Merged dataset

Dropping some links according to Amin's outlier RF Links analysis based on triangulation.

- Dropping some NA values
- Normalizing bbe and unavail_second
- Upsampling and downsampling based on the month

In [ ]:
```
mean_per_mlid = {mlid:merged_df[merged_df['mlid'] == mlid]['bbe'].mean() for mlid in list(set(merged_df['mlid']))}
std_per_mlid = {mlid:merged_df[merged_df['mlid'] == mlid]['bbe'].std() for mlid in list(set(merged_df['mlid']))}
merged_df['bbe_normalized'] = [(x - mean_per_mlid[mlid])/std_per_mlid[mlid] if std_per_mlid[mlid] > 0 else 0 for x, mlid in zip(merged_df['bbe'], merg
```

In [ ]:
```
mean_per_mlid = {mlid:merged_df[merged_df['mlid'] == mlid]['unavail_second'].mean() for mlid in list(set(merged_df['mlid']))}
std_per_mlid = {mlid:merged_df[merged_df['mlid'] == mlid]['unavail_second'].std() for mlid in list(set(merged_df['mlid']))}
merged_df['unavail_second_normalized'] = [(x - mean_per_mlid[mlid])/std_per_mlid[mlid] if std_per_mlid[mlid] > 0 else 0 for x, mlid in zip(merged_df['
```

Adding the month

In [ ]:
```
merged_df['month'] = [pd.Timestamp(x).month for x in merged_df['datetime']]
```

Handling mistaken entries (Added by JSP)[To remove scalibility score = date].

In [ ]:
```
merged_df
```

Out[ ]:

	type	datetime	tip	mlid	mw_connection_no	site_id	polarization	card_type	adaptive_modulation	freq_band	link_length	severaly_error_second	error_second	un
0	ENK	2020-01-03	NEAR	R0AP	314493.0	RL_9;SIP	Vertical	cardtype4	Enable	f3	2820.0	0.0	0.0	
1	ENK	2020-01-03	NEAR	R3XD	335068.0	RL_I7Z?H	Vertical	cardtype1	Enable	f4	1873.0	0.0	0.0	
2	ENK	2020-01-03	FAR	R0DK	306403.0	RL_I;YNL	Vertical	cardtype4	Enable	f4	1562.0	0.0	0.0	
3	ENK	2020-01-03	FAR	R0ED	313790.0	RL_I;YNL	Vertical	cardtype1	Enable	f3	3203.0	0.0	0.0	
4	ENK	2020-01-03	FAR	R0GM	1377555.0	RL_I;YNL	Vertical	cardtype1	Enable	f3	4866.0	0.0	0.0	

NA values

In [ ]:
```
# Check for NA values
merged_df.columns[merged_df.isnull().any()]
```

Out[ ]:
```
Index(['freq_band', 'scalibility_score', 'humidity_max_day1',
       'humidity_min_day1', 'wind_dir_day1', 'wind_speed_day1',
       'history_freq_band', 'history_scalibility_score',
       'history_clutter_class'],
      dtype='object')
```

In [ ]:
```
# 2442 entries are dropped #Changed to 460 when JSP was testing
merged_df['freq_band'] = merged_df['freq_band'].fillna('None')
merged_df['history_freq_band'] = merged_df['history_freq_band'].fillna('None')
merged_df['history_clutter_class'] = merged_df['history_clutter_class'].fillna('None')
merged_df['polarization'] = merged_df['polarization'].fillna('None')
merged_df['history_polarization'] = merged_df['history_polarization'].fillna('None')
merged_df['scalibility_score'] = merged_df['scalibility_score'].fillna(-1)
merged_df['history_scalibility_score'] = merged_df['history_scalibility_score'].fillna(-1)
df = merged_df.copy() #df = merged_df.dropna()
print(merged_df.shape[0] - df.shape[0])
```

```
0
```

In [ ]:
```
df.head()
```

Out[ ]:

	type	datetime	tip	mlid	mw_connection_no	site_id	polarization	card_type	adaptive_modulation	freq_band	link_length	severaly_error_second	error_second	unavail_se
0	ENK	2020-01-03	NEAR	R0AP	314493.0	RL_9;SIP	Vertical	cardtype4	Enable	f3	2820.0	0.0	0.0	
1	ENK	2020-01-03	NEAR	R3XD	335068.0	RL_I7Z?H	Vertical	cardtype1	Enable	f4	1873.0	0.0	0.0	
2	ENK	2020-01-03	FAR	R0DK	306403.0	RL_I;YNL	Vertical	cardtype4	Enable	f4	1562.0	0.0	0.0	
3	ENK	2020-01-03	FAR	R0ED	313790.0	RL_I;YNL	Vertical	cardtype1	Enable	f3	3203.0	0.0	0.0	
4	ENK	2020-01-03	FAR	R0GM	1377555.0	RL_I;YNL	Vertical	cardtype1	Enable	f3	4866.0	0.0	0.0	

Dealing with month's distribution

In [ ]:
```
# Months distribution
print(pd.DataFrame.from_dict(Counter(df['month']), orient='index').sort_values(by=0))
month_dist = dict(Counter(df['month']))
print(month_dist)
```

```
      0
1  22558
{1: 22558}
```

In [ ]:
```
'''
* Upsample: December [12]
* Downsample: July [7], September [9], October [10]
'''
from statistics import mean
middle_ground = mean([month_dist[k] for k in month_dist.keys() if k in [6,11,8]])
middle_ground = int(round(middle_ground, -1))
print(middle_ground)
```

```
7880
```

In [ ]:
```
# Downsampling July [7], September [9], October [10]
sampled_df = copy.deepcopy(df.loc[[x in [6, 8, 11] for x in df['month']]])
for m in [7, 9, 10]:
  temp = df.loc[df['month'] == m]
  temp = temp.sample(middle_ground)
  sampled_df = pd.concat([sampled_df, temp], axis = 0)
```

In [ ]:
```
# Upsampling December [12] - NAIVE APPROACH
december = df.loc[df['month'] == 12]
temp = december.sample(n=middle_ground, replace=True)
sampled_df = pd.concat([sampled_df, temp], axis = 0)
```

In [ ]:
```
Counter(sampled_df['month'])
```

Out[ ]:
```
Counter({6: 7414, 7: 7880, 8: 8000, 9: 7880, 10: 7880, 11: 8225, 12: 7880})
```

Data transformation

- We have a regression model so we decided to Normalize numerical columns such as “bbe” and “unvail_seconds” by using standard scalling
- We have a binary classifier so we decided to use One Hot Encoding for categorical columns such as “history_polarization” and “adaptive_modulation”

In [ ]:
```
# One hot encoding
columns_to_ohe = ['type', 'tip', 'polarization', 'card_type', 'adaptive_modulation', 'freq_band', 'modulation',
                  'history_type', 'history_tip', 'history_polarization', 'history_card_type', 'history_adaptive_modulation', 'history_freq_band', 'his
                  'history_clutter_class']
# df = copy.deepcopy(sampled_df) # Edited by JSP
df = copy.deepcopy(df)
for c in columns_to_ohe:
  temp = pd.get_dummies(df[c], prefix=c).astype('int')
  df = df.drop(columns=c)
  df = pd.concat([df, temp], axis=1)
```

In [ ]:
```
# df.to_csv('data_v2.csv', index=False) # Edited by JSP
df.to_csv('PREDICTION_NoSampling.csv', index=False)
```

Feature engineering

- **Date/Time feature (Year / Month / Day)**
- **Aggregation feature (sum/count/max/min)**
- **Polynomial feature**

Feature selection

- **Select feature by importance**
- **Select feature by k-best**
- **Select feature by k-square**
- **Select feature by k-scoring**

Modeling

Decision tree classifier

Import the Model

In [ ]:
```
import joblib
dtree_clf = joblib.load('20201029_dtree_clf_v4_NoSampling')
pruned_tree_features = joblib.load('20201029_dtree_clf_v4_NoSampling_inputfeatures')
```

Processing the data for the model

In [ ]:
```
to_keep = pruned_tree_features.tolist()
to_keep.append('rlf')

test_df = df.loc[:,to_keep]

# Select the target
target = 'rlf'
```

Out[ ]:

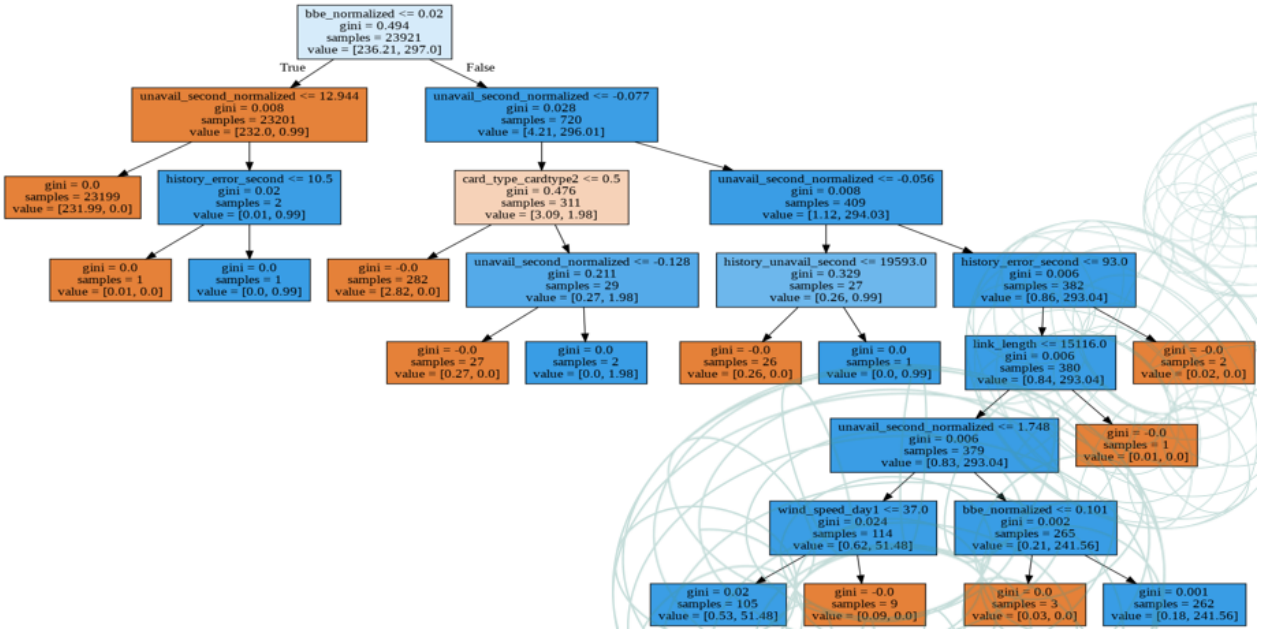
	rxlevmax	capacity	temp_max_day1	temp_min_day1	humidity_max_day1	humidity_min_day1	wind_dir_day1	wind_speed_day1	wd1_few clouds	wd1_heavy rain	wd1_heavy rain showers	wd1_l thunders witl sh
19800	-28.2	456.0	14.0	5.0	71	43	349	15	0	1	0	
19802	-39.5	160.0	14.0	5.0	71	43	349	15	0	1	0	
19872	-30.0	456.0	14.0	5.0	71	43	349	15	0	1	0	
19904	-33.3	247.0	14.0	5.0	71	43	349	15	0	1	0	
19953	-22.5	200.0	17.0	6.0	82	39	310	11	0	0	0	

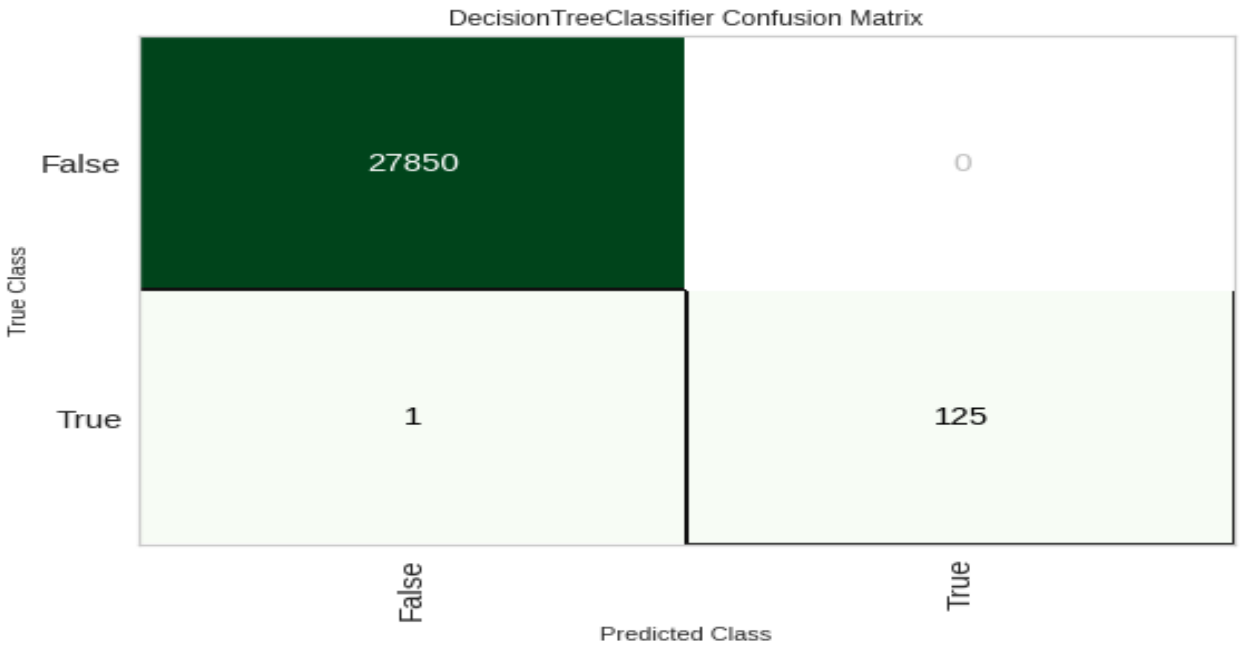
In [ ]:
```
x_pred = test_df.loc[:, test_df.columns != target].values
x_pred = np.array(x_pred).astype(float)
```

In [ ]:
```
# Predictions
list(zip(df['mlid'].values,dtree_clf.predict(x_pred)))
# test_df['mlid'].values
# ,
```

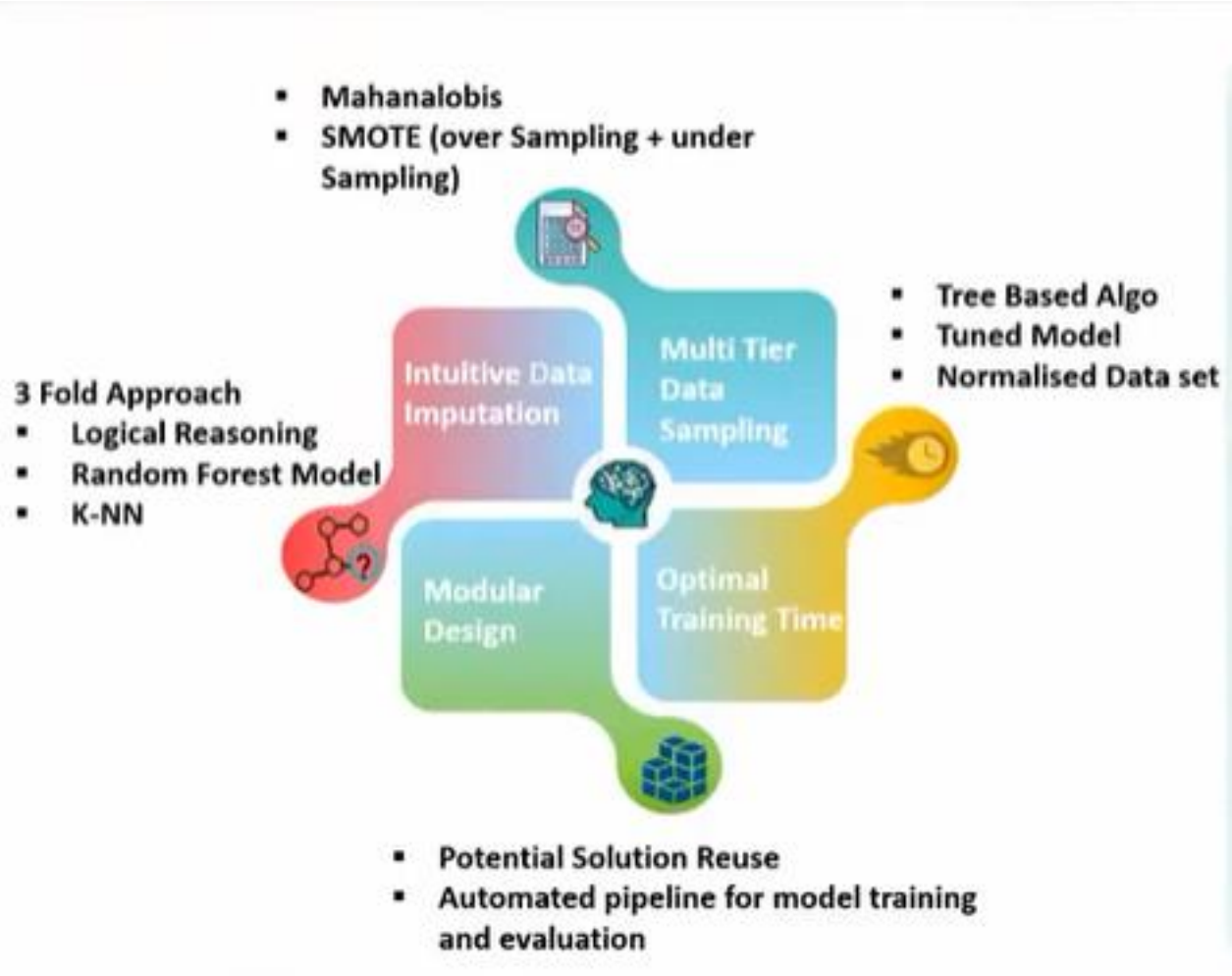
Out[ ]:
```
[('R2LC', 0.0),
 ('R2BG', 0.0),
 ('R1CR', 0.0),
 ('R0DF', 0.0),
 ('R3LM', 0.0),
```

Decision tree



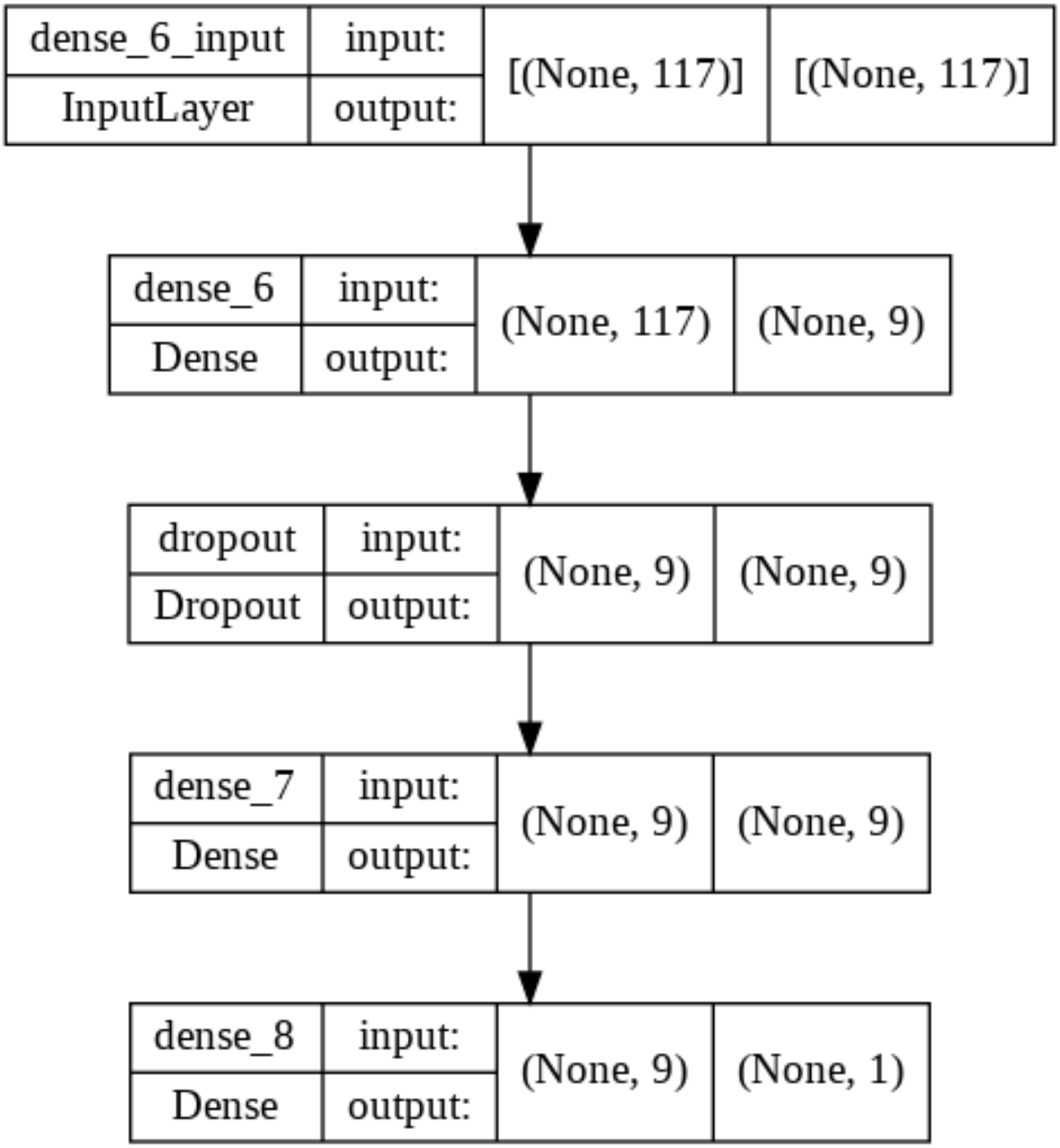


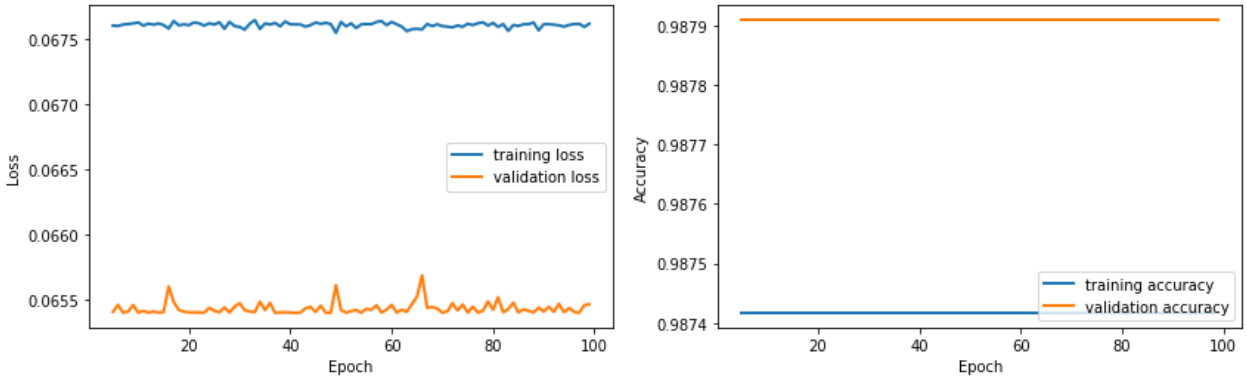
Best Model Results	
F1-Score	99.74%
Accuracy	99.7%
Model	Extra Tree



- RLF can be predicted with good reliability using weather coditions, RL KPI metrics and terrain characteristics
- Frequent re-training with new RL failure scenarios will improve model performance

Deep learning model





Machine learning vs Deep learning

	Machine learning	Deep learning
Run time	0,64 sec	0,75sec
Accuracy	99,7%	98,7%

Deployment

Conclusion

RLF prediction will augment self healing capability of networks by enabling them to take further actions to avoid service interruption / degradation.