FINAL PROJECT
MACHINE LEARNING

UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Anderson Nunes de Sousa
Daniel Rodrigues de Luna
Iago Diógenes do Rêgo
José Martins Castro Neto

1 Introduction

Case study and objectives

VANET (VEHICULAR AD HOC NETWORK)

Demands for communication on the move.

Autonomous & self configured network.

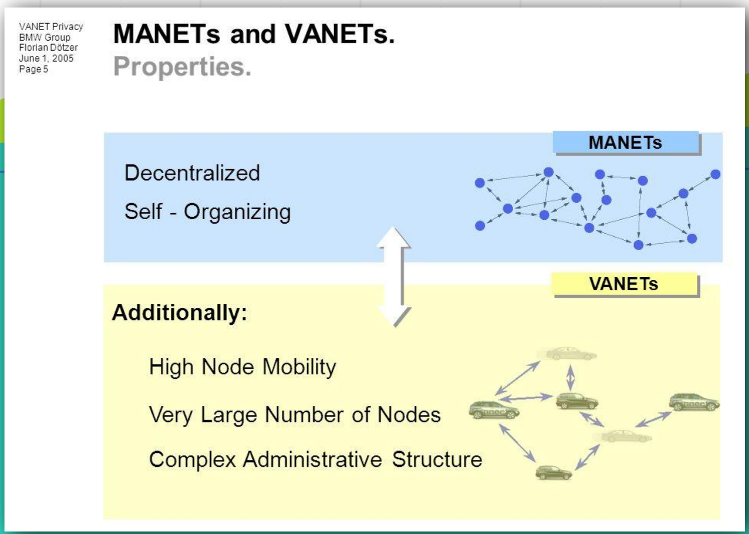
Vehicle to vehicle.

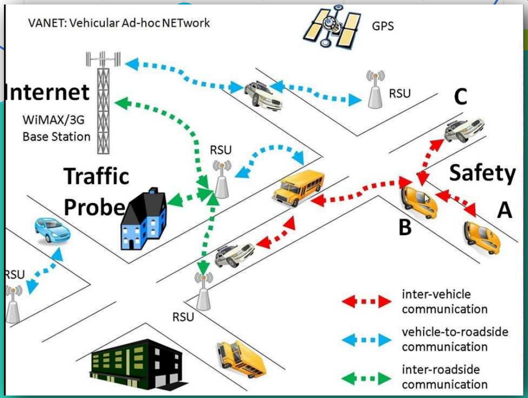
Vehicle to roadside.

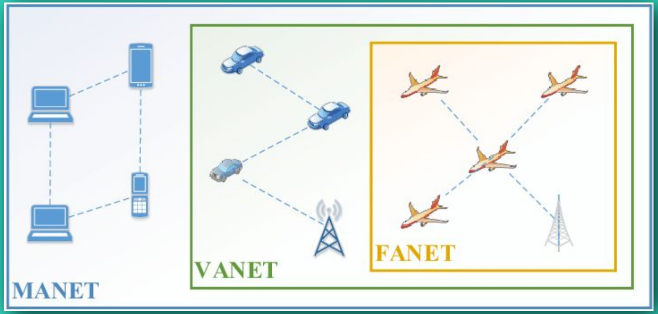
Emergency warning.

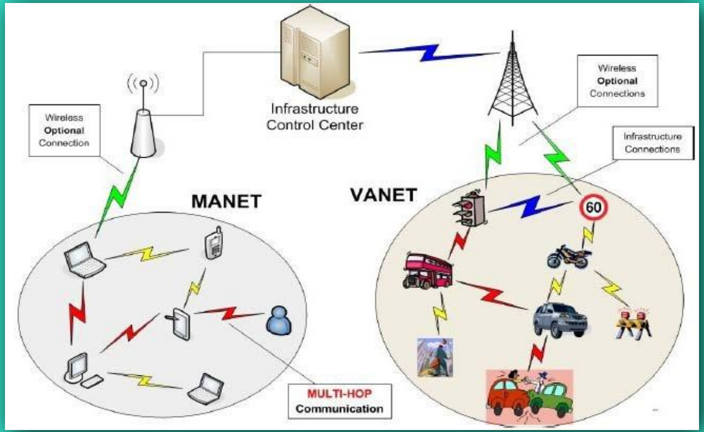
Safety.

Protocols.


VANET Privacy
BMW Group
Florian Dötzer
June 1, 2005
Page 5
MANETs and VANETs.
Properties.
MANETs
Decentralized
Self - Organizing
VANETs
Additionally:
High Node Mobility
Very Large Number of Nodes
Complex Administrative Structure


VANET: Vehicular Ad-hoc NETwork
GPS
Internet
WiMAX/3G
Base Station
RSU
C
Traffic
Probe
Safety
A
B
RSU
RSU
RSU
inter-vehicle communication
vehicle-to-roadside communication
inter-roadside communication


MANET
VANET
FANET


Wireless Optional Connection
Infrastructure Control Center
Wireless Optional Connections
Infrastructure Connections
MANET
VANET
60
MULTI-HOP Communication

ns-3

Network Simulator

ns-3 is a discrete-event network simulator for Internet systems, targeted primarily for research and educational use. ns-3 is free software, licensed under the GNU GPLv2 license, and is publicly available for research, development, and use.

ns-3

https://www.nsnam.org

All Apps

Categories

- Routing
- MANET
- Underwater Acoustic Networks
- LTE
- Public Safety
- Reinforcement Learning

more »

Newest Releases

Get Started with the App Store »

ns3-gym: OpenAI Gym integration
The Playground for Reinforcement Learning in Networking Research

mmWave Cellular Network Simulator
A millimeter wave module that can be used to evaluate cross-layer

Public Safety Communications
Models to support applications and scenarios for first responders

Epidemic Routing
Epidemic Routing for Partially-Connected Ad Hoc Networks

ns-3 App Store

https://apps.nsnam.org

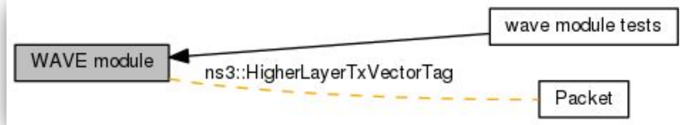
- ▶ visualizer
- ▼ wave
  - ▼ examples
    - ▶ vanet-routing-compare.cc
    - ▶ wave-simple-80211p.cc
    - ▶ wave-simple-device.cc

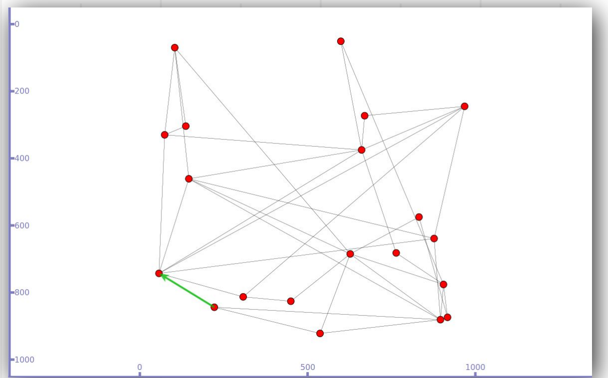
vanet-routing-compare.cc File Reference

```
#include <fstream>
#include <iostream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/aodv-module.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"
#include "ns3/dsr-module.h"
#include "ns3/applications-module.h"
#include "ns3/itu-r-1411-los-propagation-loss-model.h"
#include "ns3/ocb-wifi-mac.h"
#include "ns3/wifi-80211p-helper.h"
#include "ns3/wave-mac-helper.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/config-store-module.h"
#include "ns3/integer.h"
#include "ns3/wave-bsm-helper.h"
#include "ns3/wave-helper.h"
#include "ns3/yans-wifi-helper.h"
```

▼ WAVE module

- ▶ wave module tests
- ▶ BsmApplication
- ▶ ChannelCoordinationListener
- ▶ ChannelCoordinator
- ▶ ChannelManager
- ▶ ChannelScheduler
- ▶ ConfigStoreHelper
- ▶ CoordinationListener
- ▶ DefaultChannelScheduler
- ▶ EdcaParameter
- ▶ HigherLayerTxVectorTag
- ▶ NqosWaveMacHelper





ns-3 Network Simulator

WAVE stands for Wireless Access in Vehicular Environment.

Feature	Description	Values
'SimulationSecond'	Time simulated	Default: 300.01
'ReceiveRate'	Received Rate	Simulation Result
'PacketsReceived'	Received Packets	Simulation Result
'RoutingProtocol'	Routing Protocol	Default: AODV
'WavePktsSent'	WAVE packets sent	Default: Broadcast 10 packets
'WavePtksReceived'	WAVE packets received	Simulation Result
'WavePktsPpr'	WAVE Packet Delivery Ratio	$\frac{received}{sent}$
'ExpectedWavePktsReceived'	Expected WAVE packets Received	WavePktsSent*{n}Nodes
'ExpectedWavePktsInCoverageReceived'	Expeceted WAVE packets Received within Range	WavePktsSent*{n}Nodes within range
'BSM_PDR1'	Packet Delivery Ratio of Basic Safety Message of Range 50	Simulation Result
'BSM_PDR2'	Packet Delivery Ratio of Basic Safety Message of Range 100	Simulation Result
'BSM_PDR3'	Packet Delivery Ratio of Basic Safety Message of Range 150	Simulation Result
'BSM_PDR4'	Packet Delivery Ratio of Basic Safety Message of Range 200	Simulation Result
'BSM_PDR5'	Packet Delivery Ratio of Basic Safety Message of Range 250	Simulation Result
'BSM_PDR6'	Packet Delivery Ratio of Basic Safety Message of Range 300	Simulation Result
'BSM_PDR7'	Packet Delivery Ratio of Basic Safety Message of Range 350	Simulation Result
'BSM_PDR8'	Packet Delivery Ratio of Basic Safety Message of Range 400	Simulation Result
'BSM_PDR9'	Packet Delivery Ratio of Basic Safety Message of Range 450	Simulation Result
'BSM_PDR10'	Packet Delivery Ratio of Basic Safety Message of Range 500	Simulation Result
'MacPhyOverhead'	It is the overhead of both physical and MAC layer in IEEE 802.11	Simulation Result
'm_nodeSpeed'	Node Speed (Vehicular Speed)	Default: 20m/s

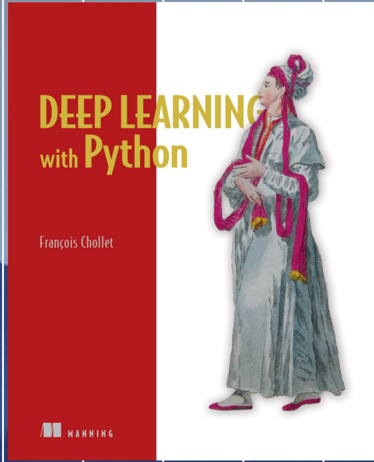
GENERATED DATASET

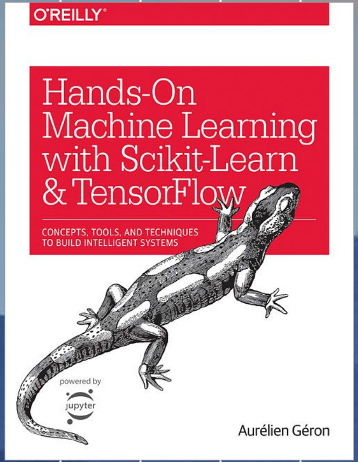
The 23 features are network related, not node related.

WE NEED TO SOLVE REAL IP/MAC PROBLEMS

- QoS (**Quality of Service**).
- **Minimum requirements** for a service to work without any major problems.

HOW MACHINE LEARNING CAN HELP US?





CLASSIFICATION

Predict the right range of velocity from the analysed node.

REGRESSION

Predict the right number of nodes of the network.

2 Methods

and Discussion

1- FIRST ANALYSIS

MAIN PARAMETERS:

Parameters	Analysis 1	Analysis 2
'SimulationSecond'	1000.01	1000.01
'Routing Protocol'	AODV	AODV
'Propagation Model'	TwoRayGround	TwoRayGround
'PHY Layer'	802.11p	802.11p
'Number of Nodes'	Default (156)	{10, 20, ... , 200}
'Scenario'	Grid 300 m x 1500 m	Grid 300 m x 1500 m
'Node Speed'	{6, 12, 24, 33} m/s	Default (20 m/s)

1.1- IMPORTING DATASETS

```
[ ]  # Importing 4 datasets

     # Upload do dataset
     from google.colab import files

     uploaded = files.upload()

     for fn in uploaded.keys():
       print('User uploaded file "{name}" with length {length} bytes'.format(
           name=fn, length=len(uploaded[fn])))
```

```
User uploaded file "vanet-routing.output_speed6.csv" with length 153339 bytes
User uploaded file "vanet-routing.output_speed12.csv" with length 154626 bytes
User uploaded file "vanet-routing.output_speed24.csv" with length 154677 bytes
User uploaded file "vanet-routing.output_speed33.csv" with length 154681 bytes
```

```
[ ]  #Library (to be completed)
     import pandas as pd
     import numpy as np
     import seaborn as sns
```

```
[ ]  wave6 = pd.read_csv("vanet-routing.output_speed6.csv",",")
     wave12 = pd.read_csv("vanet-routing.output_speed12.csv",",")
     wave24 = pd.read_csv("vanet-routing.output_speed24.csv",",")
     wave33 = pd.read_csv("vanet-routing.output_speed33.csv",",")
```

1.2- DATA PROCESSING

```
wave6.head()
```

	SimulationSecond	ReceiveRate	PacketsReceived	NumberOfSinks	RoutingProtocol
0	0	0.000	0	10	protocol
1	1	0.000	0	10	protocol
2	2	5.120	10	10	protocol
3	3	13.824	27	10	protocol
4	4	19.968	39	10	protocol

5 rows × 23 columns

```
#Chaging RoutingProtocol column for the right name

wave6['RoutingProtocol'] = 'AODV'
wave12['RoutingProtocol'] = 'AODV'
wave24['RoutingProtocol'] = 'AODV'
wave33['RoutingProtocol'] = 'AODV'
```

```
dframes = [wave6, wave12, wave24, wave33]
wave = pd.concat(dframes, ignore_index=True)
```

```
print(wave.shape)
print(wave.isnull().sum())
```

```
(4004, 23)
SimulationSecond                    0
ReceiveRate                         0
PacketsReceived                     0
NumberOfSinks                       0
RoutingProtocol                     0
TransmissionPower                   0
WavePktsSent                        0
WavePtksReceived                    0
WavePktsPpr                         0
ExpectedWavePktsReceived            0
ExpectedWavePktsInCoverageReceived  0
BSM_PDR1                            0
BSM_PDR2                            0
BSM_PDR3                            0
BSM_PDR4                            0
BSM_PDR5                            0
BSM_PDR6                            0
BSM_PDR7                            0
BSM_PDR8                            0
BSM_PDR9                            0
BSM_PDR10                           0
MacPhyOverhead                      0
m_nodeSpeed                         0
dtype: int64
```

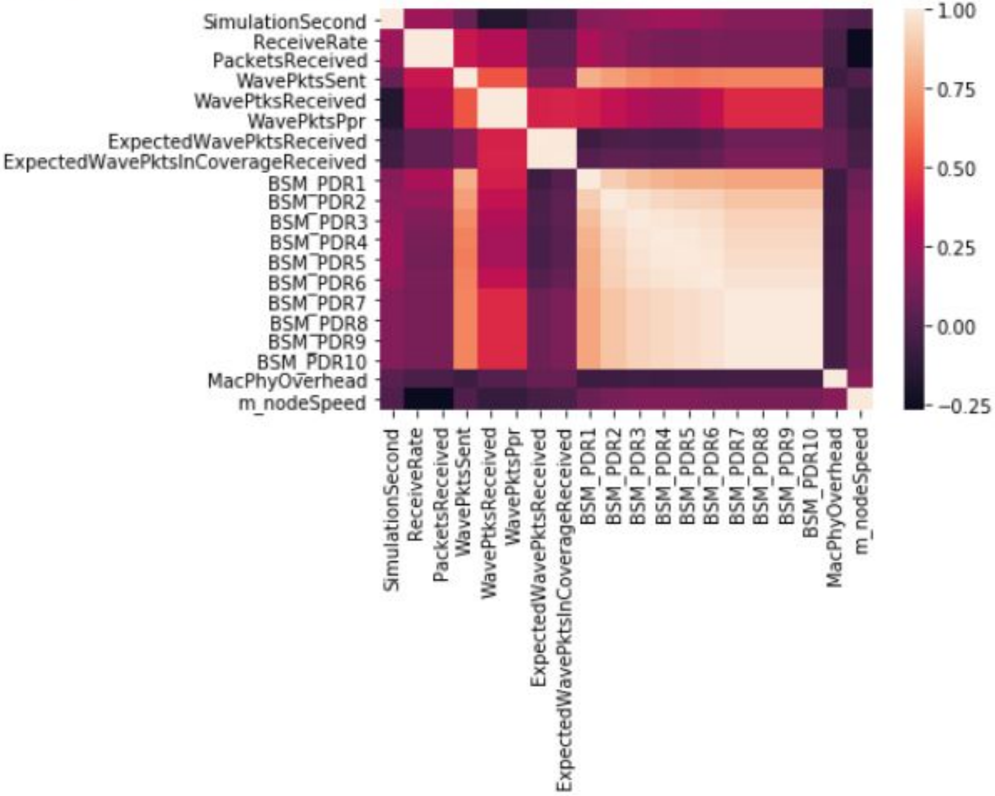
```
#shuffle index
wave=wave.sample(frac=1)
wave=wave.reset_index(drop=True)
wave.head()
```

	SimulationSecond	ReceiveRate	PacketsReceived	NumberOfSinks	RoutingProtocol	TransmissionPower	WavePktsSent	WavePtksReceived
0	767	20.480	40	10	AODV	7.5	400	5230
1	792	16.384	32	10	AODV	7.5	400	5097
2	20	19.968	39	10	AODV	7.5	400	5389
3	521	20.480	40	10	AODV	7.5	400	4586
4	192	15.360	30	10	AODV	7.5	400	5396

5 rows × 23 columns

```
wave = wave.drop(['TransmissionPower', 'NumberOfSinks'],axis=1)
sns.heatmap(wave.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x276b68ea978>
```



MinMax Scaler

Transforms features by scaling each feature to a given range.

```
cut_points = [-1,10,20,30,40]
label_names = ["6m/s","12m/s","24m/s","33m/s"]
wave["speed_categories"] = pd.cut(wave["m_nodeSpeed"],cut_points,labels=label_names)

#sns.pairplot(wave,hue="speed_categories")
```

```
from sklearn.preprocessing import MinMaxScaler

features=['SimulationSecond','ReceiveRate','PacketsReceived','WavePktsSent','WavePtksReceived','WavePktsPpr',
        'ExpectedWavePktsInCoverageReceived','BSM_PDR1','BSM_PDR2','BSM_PDR3','BSM_PDR4','BSM_PDR5','BSM_PDR6
        'MacPhyOverhead','m_nodeSpeed']

for feature in features:
  scaler = MinMaxScaler()
  wave[[feature]]=scaler.fit_transform(wave[[feature]])
```

```
wave.head()
```

	SimulationSecond	ReceiveRate	PacketsReceived	RoutingProtocol	WavePktsSent	WavePtksReceived	Wav
0	0.767	0.869565	0.869565	AODV	1.0	0.825181	
1	0.792	0.695652	0.695652	AODV	1.0	0.804197	
2	0.020	0.847826	0.847826	AODV	1.0	0.850268	
3	0.521	0.869565	0.869565	AODV	1.0	0.723572	
4	0.192	0.652174	0.652174	AODV	1.0	0.851373	

5 rows × 22 columns

```
wave.columns
```

```
Index(['SimulationSecond', 'ReceiveRate', 'PacketsReceived', 'RoutingProtocol',
       'WavePktsSent', 'WavePtksReceived', 'WavePktsPpr',
       'ExpectedWavePktsReceived', 'ExpectedWavePktsInCoverageReceived',
       'BSM_PDR1', 'BSM_PDR2', 'BSM_PDR3', 'BSM_PDR4', 'BSM_PDR5', 'BSM_PDR6',
       'BSM_PDR7', 'BSM_PDR8', 'BSM_PDR9', 'BSM_PDR10', 'MacPhyOverhead',
       'm_nodeSpeed', 'speed_categories'],
      dtype='object')
```

1.3- TRAINING AND TEST

```
#Training and test datasets
#from sklearn.cross_validation import train_test_split
#X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.4,random_state=101)

train = wave.iloc[0:np.rint(0.6*len(wave)).astype(int)]
print(len(wave))
print(train.head())
```

```
4004
  SimulationSecond  ReceiveRate  PacketsReceived RoutingProtocol  \
0            0.767     0.869565         0.869565            AODV
1            0.792     0.695652         0.695652            AODV
2            0.020     0.847826         0.847826            AODV
3            0.521     0.869565         0.869565            AODV
4            0.192     0.652174         0.652174            AODV

  WavePktsSent  WavePtksReceived  WavePktsPpr  ExpectedWavePktsReceived  \
0          1.0          0.825181     0.825181                  0.781679
1          1.0          0.804197     0.804197                  0.458015
2          1.0          0.850268     0.850268                  0.635115
3          1.0          0.723572     0.723572                  0.378626
4          1.0          0.851373     0.851373                  0.864122

  ExpectedWavePktsInCoverageReceived  BSM_PDR1       ...        BSM_PDR4  \
0                           0.817241  0.925781       ...        0.846598
1                           0.491379  0.950000       ...        0.885885
2                           0.658621  0.918269       ...        0.787870
3                           0.382759  0.895161       ...        0.838138
4                           0.853448  0.874558       ...        0.759764

   BSM_PDR5  BSM_PDR6  BSM_PDR7  BSM_PDR8  BSM_PDR9  BSM_PDR10  \
0  0.833639  0.846662  0.829334  0.829334  0.829334   0.829334
1  0.870268  0.876293  0.845720  0.845720  0.845720   0.845720
2  0.771426  0.784939  0.774071  0.774071  0.774071   0.774071
3  0.832527  0.840175  0.806496  0.806496  0.806496   0.806496
4  0.748966  0.767947  0.758540  0.758540  0.758540   0.758540

   MacPhyOverhead  m_nodeSpeed  speed_categories
0        0.461180     0.000000              6m/s
1        0.466635     0.666667             24m/s
2        0.473689     1.000000             33m/s
3        0.458177     0.000000              6m/s
4        0.469631     0.000000              6m/s

[5 rows x 22 columns]
```

1.4- SELECT THE BEST-PERFORMING FEATURES

```
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import Perceptron
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel

def select_features(df,index):

    # index
    # 0 - random forest
    # 1 - logistic regression

    # Remove non-numeric columns, columns that have null values
    #df = df.select_dtypes([np.number]).dropna(axis=1)
    all_X = df.drop(['SimulationSecond','RoutingProtocol','m_nodeSpeed', 'speed_categories'],axis=1)
    all_y = df["speed_categories"]

    clf_rf = RandomForestClassifier(random_state=1, n_estimators=100)
    clf_lr = LogisticRegression()
    clf_per = Perceptron(tol=1e-9)
    clf_sgd = SGDClassifier()
    clfs = [clf_rf,clf_lr,clf_per,clf_sgd]
```

```
#selector = RandomForestClassifier(n_estimators=50, max_features='sqrt')
#selector.fit(all_X, all_y)
#features=pd.DataFrame()
#features['feature'] = all_X.columns
#features['importance'] = selector.feature_importances_
#features.sort_values(by=['importance'],ascending=True,inplace=True)
#features.set_index('feature',inplace=True)

#model = SelectFromModel(selector, prefit=True)
#train_reduced = model.transform(all_X)

selector = RFECV(clfs[index],cv=10,n_jobs=-1)
selector.fit(all_X,all_y)
#best_columns = train_reduced.shape
best_columns = list(all_X.columns[selector.support_])
#print("Best Columns \n"+"-"*12+"\n{}\n".format(best_columns))

return best_columns
#return features
```

```
#features_importance = select_features(train,0)
#cols_rf.plot(kind='barh',figsize=(25,25))
#a = features_importance.sort_values(by=['importance'],ascending=False)
#cols_features_selected=a.iloc[0:10,0].index
cols_rf = select_features(train,0)
cols_lr = select_features(train,1)
cols_per = select_features(train,2)
cols_sgd = select_features(train,3)
print(cols_rf)
print(cols_lr)
print(cols_per)
print(cols_sgd)
```

1.5- SELECT AND TURNING DIFFERENT ALGORITHMS

```
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import Perceptron
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel

def select_features(df,index):
```

```
# Remove non-numeric columns, columns that have null values
#df = df.select_dtypes([np.number]).dropna(axis=1)
all_X = df.drop(['SimulationSecond','RoutingProtocol','m_nodeSpeed', 'speed_categorie
all_y = df["speed_categories"]

clf_rf = RandomForestClassifier(random_state=1, n_estimators=100)
clf_lr = LogisticRegression()
clf_per = Perceptron(tol=1e-9)
clf_sgd = SGDClassifier()
clfs = [clf_rf,clf_lr,clf_per,clf_sgd]
```

```
#model = SelectFromModel(selector, prefit=True)
#train_reduced = model.transform(all_X)

selector = RFECV(clfs[index],cv=10,n_jobs=-1)
selector.fit(all_X,all_y)
#best_columns = train_reduced.shape
best_columns = list(all_X.columns[selector.support_])
#print("Best Columns \n"+"-"*12+"\n{}\n".format(best_columns))

return best_columns
#return features
```

1.6- TEST SET

```
#Test set

holdout = wave.iloc[np.rint(0.6*len(wave)).astype(int):]

rf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=20, max_features='log2', 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=200, n_jobs=1,
            oob_score=False, random_state=1, verbose=0, warm_start=False)

rf.fit(train[cols_rf],train['speed_categories'])
print(rf.score(holdout[cols_rf],holdout['speed_categories']))
```

```
0.7896379525593009
```

“

We were able to get ***78,96%*** *of accuracy at test set using* ***RandomForest*** *classifier considering the "cols_rf" features and RandomForest Classifier from "results_a".*

1.7- NEW FEATURES

We dropped all original BSM_PDRs features and keeped just the ones created. Then, we passed through the same classifiers from the topic before.

```
wave.head()
bsmpdr15 = ['BSM_PDR1', 'BSM_PDR2','BSM_PDR3', 'BSM_PDR4','BSM_PDR5']
bsmpdr610 = ['BSM_PDR6', 'BSM_PDR7','BSM_PDR8', 'BSM_PDR9','BSM_PDR10']

wave_copy = wave.copy()

wave_copy['BSM_PDR1-5_MEAN'] = wave_copy[bsmpdr15].mean(axis=1)
wave_copy['BSM_PDR6-10_MEAN'] = wave_copy[bsmpdr610].mean(axis=1)
wave_copy = wave_copy.drop(['BSM_PDR1', 'BSM_PDR2','BSM_PDR3', 'BSM_PDR4',

train2 = wave_copy.iloc[0:np.rint(0.6*len(wave_copy)).astype(int)]
holdout2=wave_copy.iloc[np.rint(0.6*len(wave_copy)).astype(int):]

cols_rf = select_features(train2,0)
cols_lr = select_features(train2,1)
cols_per = select_features(train2,2)
cols_sgd = select_features(train2,3)


print(cols_rf)
print(cols_lr)
print(cols_per)
print(cols_sgd)
```

1.8- TEST SET VALIDATION

```
#test set validation
rf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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=10, n_jobs=1,
            oob_score=False, random_state=1, verbose=0, warm_start=False)

rf.fit(train2[cols_rf],train2['speed_categories'])
print(rf.score(holdout2[cols_rf],holdout2['speed_categories']))
```

```
0.742821473158518
```

“

Even creating two new features and Dropping all BSM_PDR{n} ones.

Our results did not improve.

We got **74,28%** at test set.

1.9- NEURAL NETWORKS CLASSIFIER FROM KERAS

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
```

```
#Importing the dataset again
wave6k = pd.read_csv("vanet-routing.output_speed6.csv",",")
wave12k = pd.read_csv("vanet-routing.output_speed12.csv",",")
wave24k = pd.read_csv("vanet-routing.output_speed24.csv",",")
wave33k = pd.read_csv("vanet-routing.output_speed33.csv",",")

#Combining the csv files into one dataset
dframesk = [wave6k, wave12k, wave24k, wave33k]
wavek = pd.concat(dframesk, ignore_index=True)

#Shuffle samples
wavek=wavek.sample(frac=1)
wavek=wavek.reset_index(drop=True)
wavek.head()

#Creating a column for labels
#cut_points = [-1,10,20,30,40]
#label_names = ["6m/s","12m/s","24m/s","33m/s"]
wavek["speed_categories"] = wavek["m_nodeSpeed"]
```

```
#Applying MinMaxScaler
features=['SimulationSecond','ReceiveRate','PacketsReceived','WavePktsSent','WavePtksRece
        'ExpectedWavePktsInCoverageReceived','BSM_PDR1','BSM_PDR2','BSM_PDR3','BSM_PDR4',
        'MacPhyOverhead','m_nodeSpeed','speed_categories']

for feature in features:
  scaler = MinMaxScaler()
  wavek[[feature]]=scaler.fit_transform(wavek[[feature]])

#Creating new features
bsmpdr15 = ['BSM_PDR1', 'BSM_PDR2','BSM_PDR3', 'BSM_PDR4','BSM_PDR5']
bsmpdr610 = ['BSM_PDR6', 'BSM_PDR7','BSM_PDR8', 'BSM_PDR9','BSM_PDR10']

wavek_copy = wavek.copy()

wavek_copy['BSM_PDR1-5_MEAN'] = wavek_copy[bsmpdr15].mean(axis=1)
wavek_copy['BSM_PDR6-10_MEAN'] = wavek_copy[bsmpdr610].mean(axis=1)
wavek_copy = wavek_copy.drop(['BSM_PDR1', 'BSM_PDR2','BSM_PDR3', 'BSM_PDR4','BSM_PDR5','B

#Separate train and test
train_k = wavek_copy.iloc[0:np.rint(0.6*len(wavek_copy)).astype(int)]
test_k = wavek_copy.iloc[np.rint(0.6*len(wavek_copy)).astype(int):]

#Creating training and test dataframes
X_traink = train_k.drop(['SimulationSecond','RoutingProtocol','m_nodeSpeed', 'speed_categ
y_traink = train_k["speed_categories"]

X_testk = test_k.drop(['SimulationSecond','RoutingProtocol','m_nodeSpeed', 'speed_categor
y_testk = test_k["speed_categories"]
```

```python
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import plot_model
from keras.utils import np_utils
```

```python
def plot_decision_boundary(X, y, model, steps=1000, cmap='Paired'):
    """
    Function to plot the decision boundary and data points of a model.
    Data points are colored based on their actual label.
    """
    cmap = plt.get_cmap(cmap)

    # Define region of interest by data limits
    xmin, xmax = X[:,0].min() - 0.1, X[:,0].max() + 0.1
    ymin, ymax = X[:,1].min() - 0.1, X[:,1].max() + 0.1
    steps = 1000
    x_span = np.linspace(xmin, xmax, steps)
    y_span = np.linspace(ymin, ymax, steps)
    xx, yy = np.meshgrid(x_span, y_span)

    # Make predictions across region of interest
    labels = model.predict(np.c_[xx.ravel(), yy.ravel()])

    # Plot decision boundary in region of interest
    z = labels.reshape(xx.shape)

    fig, ax = plt.subplots()
    ax.contourf(xx, yy, z, cmap=cmap, alpha=0.5)

    # Get predicted labels on training data and plot
    train_labels = model.predict(X)
    ax.scatter(X[:,0], X[:,1], c=y, cmap=cmap, lw=0)

    return fig, ax
```

```
#treating the labels

# encode class values as integers
encoder = LabelEncoder()
encoder.fit(y_traink)
encoded_Y = encoder.transform(y_traink)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y_train = np_utils.to_categorical(encoded_Y)
```

```
dummy_y_train
```

```
array([[0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       ...,
       [0., 1., 0., 0.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.]], dtype=float32)
```

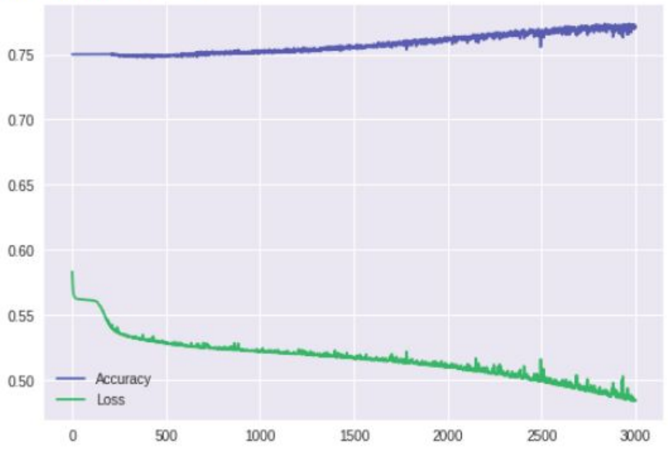
```
#treating the labels

# encode class values as integers
encoder = LabelEncoder()
encoder.fit(y_testk)
encoded_Y = encoder.transform(y_testk)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y_test = np_utils.to_categorical(encoded_Y)
```

```
dummy_y_test
```

```
array([[0., 0., 0., 1.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       ...,
       [0., 0., 1., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.]], dtype=float32)
```

```
model1 = Sequential()

model1.add(Dense(16, input_dim=12, activation='relu'))
model1.add(Dense(8, activation='relu'))
model1.add(Dense(8, activation='relu'))
model1.add(Dense(4, activation='softmax'))

model1.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

%time history1 = model1.fit(X_traink, dummy_y_train, epochs=3000,

plt.plot(history1.history['acc'],label='Accuracy')
plt.plot(history1.history['loss'],label='Loss')
plt.legend()
plt.show()
```

```
CPU times: user 1min 31s, sys: 12.2 s, total: 1min 43s
Wall time: 1min 17s
```



```
history1.history['acc'][-1]
```

```
0.7709200674052242
```

```
model_pred = model1.predict(X_testk,verbose=1)

for i in np.arange(0,len(model_pred)):
    max_index = list(model_pred[i]).index(np.amax(model_pred[i]))
    model_pred[i,max_index] = 1

count = 0
for i in range(0,len(model_pred)):
    max_index = list(model_pred[i]).index(np.amax(model_pred[i]))

    if (model_pred[i,max_index]==dummy_y_test[i,max_index]):
        count += 1

print(count)
print('Acertos: ', count/len(model_pred))
```

```
1602/1602 [==============================] - 0s 86us/step
739
Acertos:  0.4612983770287141
```

```python
model_pred
```

```
array([[0.08023544, 0.29762766, 0.28410557, 1.        ],
       [0.19741194, 1.        , 0.2410597 , 0.26794764],
       [0.1314931 , 0.30129606, 0.26181477, 1.        ],
       ...,
       [1.        , 0.26039645, 0.21340607, 0.21176995],
       [0.26433963, 1.        , 0.227215  , 0.23370676],
       [0.03844834, 0.29398775, 0.28264546, 1.        ]]
```

```python
dummy_y_test
```

```
array([[0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 1., 0., 0.],
       ...,
       [1., 0., 0., 0.],
       [0., 0., 0., 1.],
       [0., 0., 1., 0.]], dtype=float32)
```

```python
list(model_pred[0]).index(np.amax(model_pred[0]))
```

```
3
```

1.10- CONCLUSION

	TRAINING	TEST
FIRST TEST	60%	78.96%
SECOND TEST	60%	74.28%
THIRD TEST	60%	77%

2- SECOND ANALYSIS

Predict the right number of nodes (**n_nodes**) of the network.

IMPORTING DATASETS

- 19 simulations
- Nodes from 10 to 200 (step=10)
- 1000 samples for each simulation
- Final dataset has 19000 samples

```
vanet = pd.read_csv("vanet-routing.output_node_all.csv")
vanet.head()
```

	SimulationSecond	ReceiveRate	PacketsReceived	NumberOfSinks	RoutingProtocol
0	459	16.384	32	10	protocol
1	582	19.968	39	10	protocol
2	708	12.288	24	10	protocol
3	396	17.408	34	10	protocol
4	699	16.384	32	10	protocol

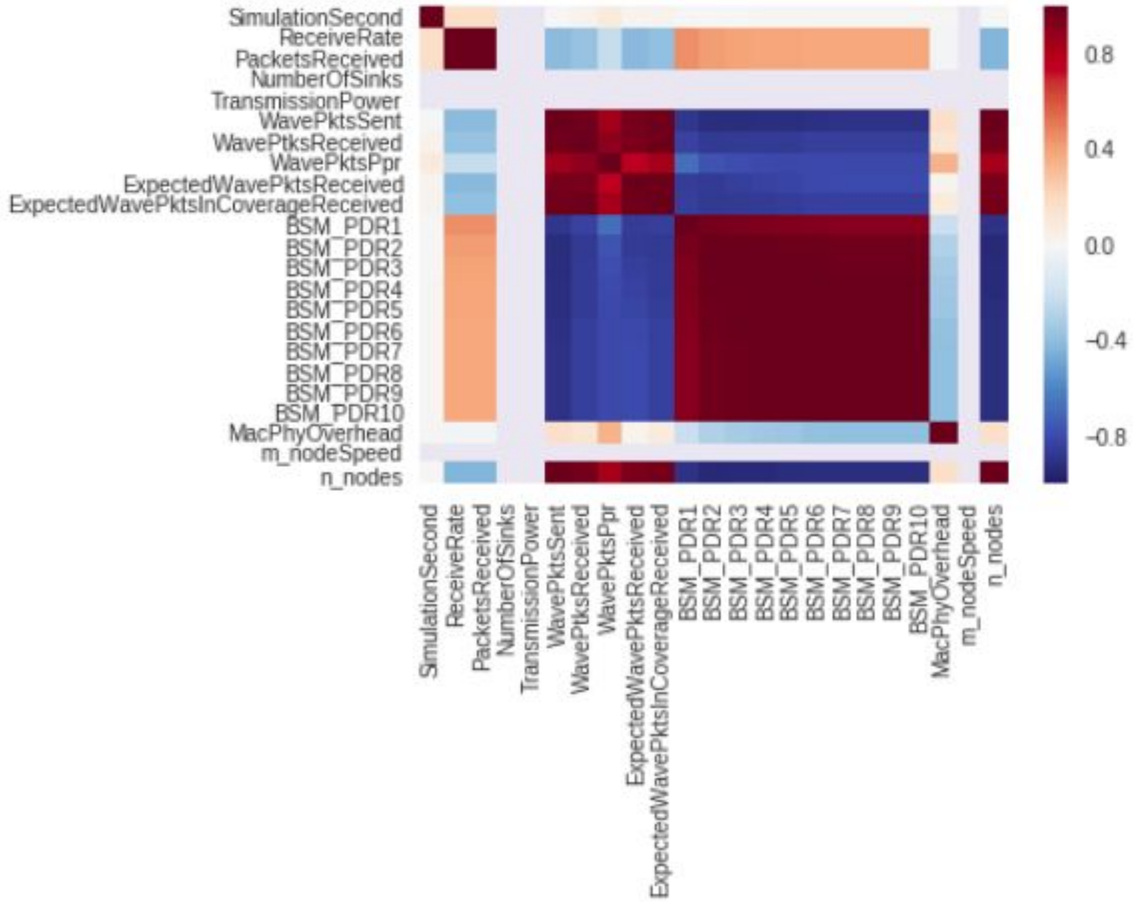
5 rows × 24 columns

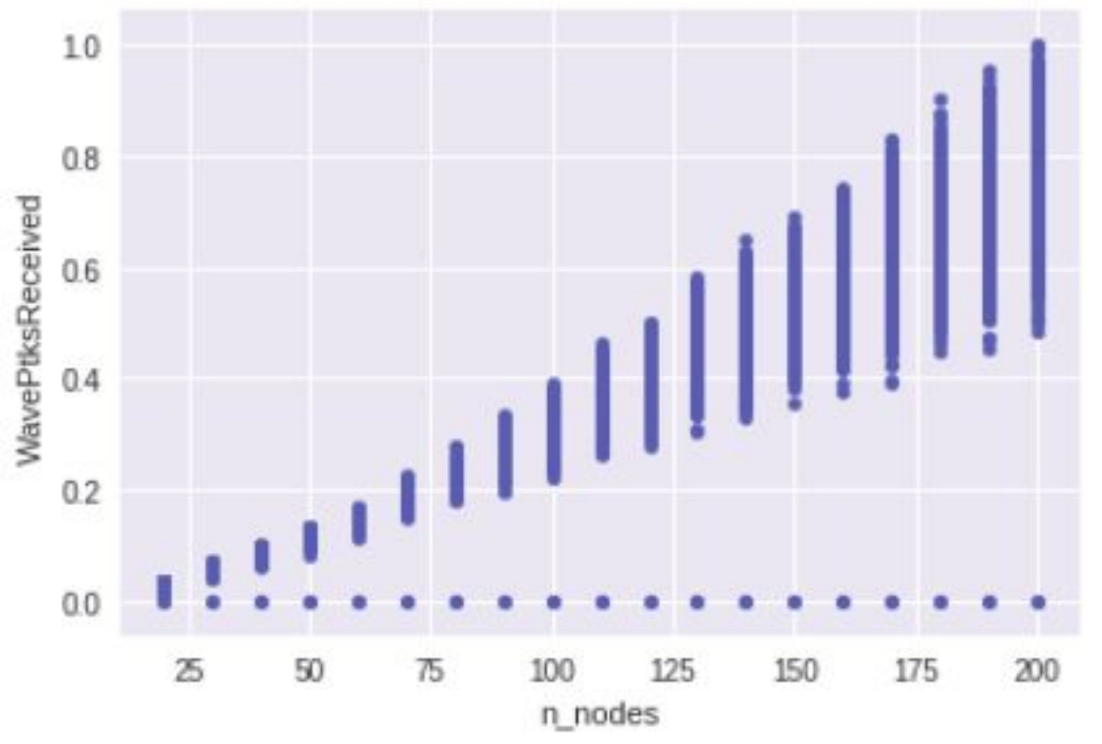
```python
from sklearn.preprocessing import MinMaxScaler

features=['SimulationSecond','ReceiveRate','PacketsReceived','WavePktsSent','WavePtksRece
        'ExpectedWavePktsInCoverageReceived','BSM_PDR1','BSM_PDR2','BSM_PDR3','BSM_PDR4',
        'MacPhyOverhead','m_nodeSpeed']

for feature in features:
  scaler = MinMaxScaler()
  vanet[[feature]]=scaler.fit_transform(vanet[[feature]])
```

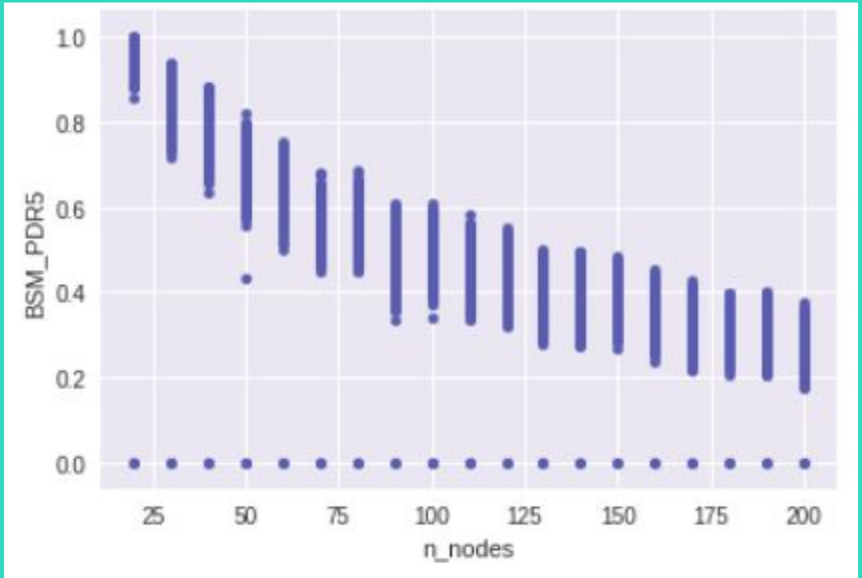
```python
train = vanet.iloc[0:np.rint(0.6*len(vanet)).astype(int)]
print(len(vanet))
print(train.head())

holdout = vanet.iloc[np.rint(0.6*len(vanet)).astype(int):]
```

```
19019
   SimulationSecond  ReceiveRate  PacketsReceived  NumberOfSinks  \
0             0.459     0.561404         0.561404             10
1             0.582     0.684211         0.684211             10
2             0.708     0.421053         0.421053             10
3             0.396     0.596491         0.596491             10
4             0.699     0.561404         0.561404             10
```

```
   WavePktsPpr  ExpectedWavePktsReceived  ...   BSM_PDR4  BSM_PDR5  \
0     0.528254                  0.108271  ...   0.541032  0.513870
1     0.711323                  0.346313  ...   0.418215  0.392977
2     0.628511                  0.342428  ...   0.385750  0.351837
3     0.590424                  0.310654  ...   0.390144  0.353879
4     0.881266                  0.826714  ...   0.334789  0.312529


   BSM_PDR6  BSM_PDR7  BSM_PDR8  BSM_PDR9  BSM_PDR10  MacPhyOverhead
0  0.487150  0.473090  0.473090  0.473090   0.473090        0.510912
1  0.368122  0.349492  0.349492  0.349492   0.349492        0.504567
2  0.321372  0.300136  0.300136  0.300136   0.300136        0.501674
3  0.321337  0.300003  0.300003  0.300003   0.300003        0.494955
4  0.290229  0.273366  0.273366  0.273366   0.273366        0.473248

   m_nodeSpeed  n_nodes
0          0.0     70.0
1          0.0    130.0
2          0.0    140.0
3          0.0    140.0
4          0.0    200.0
```


SimulationSecond
ReceiveRate
PacketsReceived
NumberOfSinks
TransmissionPower
WavePktsSent
WavePtksReceived
WavePktsPpr
ExpectedWavePktsReceived
ExpectedWavePktsInCoverageReceived
BSM_PDR1
BSM_PDR2
BSM_PDR3
BSM_PDR4
BSM_PDR5
BSM_PDR6
BSM_PDR7
BSM_PDR8
BSM_PDR9
BSM_PDR10
MacPhyOverhead
m_nodeSpeed
n_nodes
0.8
0.4
0.0
-0.4
-0.8


1.0
0.8
0.6
0.4
0.2
0.0
WavePtksReceived
25
50
75
100
125
150
175
200
n_nodes



- WavePtksSent
- WavePtsReceived
- WavePtksPpr
- ExpectedWavePtksReceveid
- ExpectedWavePtksInCoverageReceveid

```python
from sklearn.feature_selection import RFECV
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.neural_network import MLPRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import Perceptron

def select_features(df,index):


    all_X = df.drop(['SimulationSecond','RoutingProtocol','TransmissionPower', 'NumberOfSinks','n_nodes'
    all_y = df["n_nodes"]

    clf_lr = LinearRegression()
    clf_la = Lasso(alpha=0.1)
    clf_ri = Ridge(alpha=0.1)
    clf_sgd = SGDRegressor(max_iter=1000, penalty=None, eta0=0.1, random_state=42)

    clfs = [clf_lr,clf_la,clf_ri, clf_sgd]
```

```
    selector = RFECV(clfs[index],cv=10,n_jobs=-1)
    selector.fit(all_X,all_y)
    best_columns = list(all_X.columns[selector.support_])

    return best_columns

cols_lr = select_features(train,0)
cols_la = select_features(train,1)
cols_ri = select_features(train,2)
cols_sgd = select_features(train,3)

print(cols_lr)
print(cols_la)
print(cols_ri)
print(cols_sgd)
```

```
clf_lr = LinearRegression()
clf_la = Lasso(alpha=0.1)
clf_ri = Ridge(alpha=0.1)
clf_sgd = SGDRegressor(max_iter=1000, penalty=None, eta0=0.1, random_state=42)
clf_NNR = MLPRegressor(max_iter=1000)
```

```
clf_lr.fit(train[cols_lr],train["n_nodes"])
clf_la.fit(train[cols_la],train["n_nodes"])
clf_ri.fit(train[cols_ri],train["n_nodes"])
clf_sgd.fit(train[cols_sgd],train["n_nodes"])
clf_NNR.fit(train[cols_lr],train["n_nodes"])
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:183:
  FutureWarning)
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=1000, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
```

```python
pred_lr = clf_lr.predict(holdout[cols_lr])
pred_la = clf_la.predict(holdout[cols_la])
pred_ri = clf_ri.predict(holdout[cols_ri])
pred_sgd = clf_sgd.predict(holdout[cols_sgd])
pred_NNR = clf_NNR.predict(holdout[cols_lr])
```

```python
rmse_lr = np.sqrt(mean_squared_error(pred_lr, holdout["n_nodes"]))
rmse_la = np.sqrt(mean_squared_error(pred_la, holdout["n_nodes"]))
rmse_ri = np.sqrt(mean_squared_error(pred_ri, holdout["n_nodes"]))
rmse_sgd = np.sqrt(mean_squared_error(pred_sgd, holdout["n_nodes"]))
rmse_NNR = np.sqrt(mean_squared_error(pred_NNR, holdout["n_nodes"]))
```

```python
print(rmse_lr)
print(rmse_la)
print(rmse_ri)
print(rmse_sgd)
print(rmse_NNR)
```

```
3.027190478710573
3.3589923958656938
3.043837302669662б
3.0635483855654564
2.076774985097809
```

```python
print(pred_NNR[0:10])
print("--------------")
print(holdout["n_nodes"].head(10).values)
```

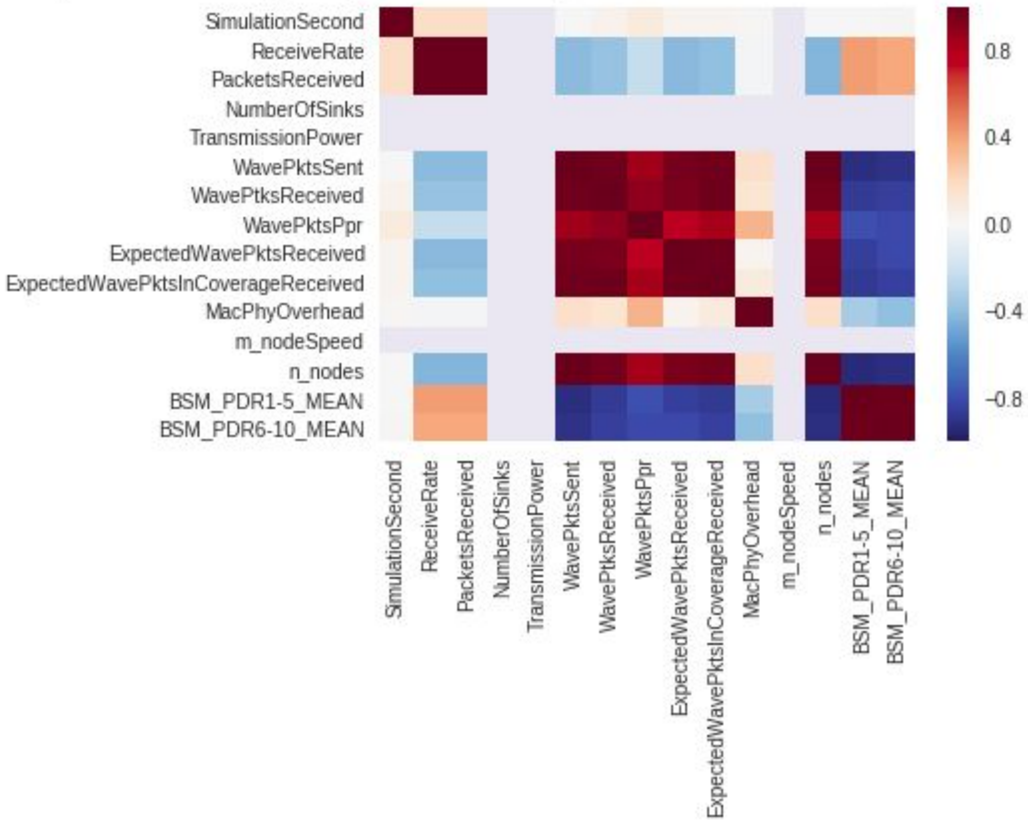
```
[ 29.1702104   20.50486322 200.23944072 149.93595134 150.52449009
 169.94018423 179.98630155  90.86888499 100.31288586  40.36854123]
--------------
[ 30.  20. 200. 150. 150. 170. 180.  90. 100.  40.]
```

```
pred_NNR_train = clf_NNR.predict(train[cols_lr])
rmse_NNR_train = np.sqrt(mean_squared_error(pred_NNR_train, train["n_nodes"]))
print(rmse_NNR_train)
```

2.748336593123354

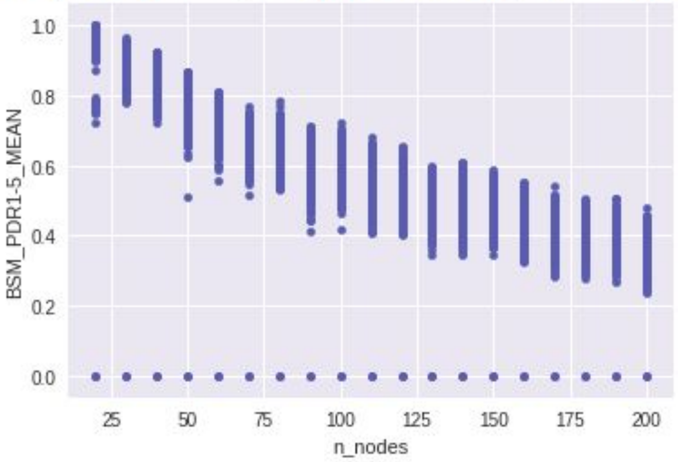
```
[ ]  bsmpdr15 = ['BSM_PDR1', 'BSM_PDR2','BSM_PDR3', 'BSM_PDR4','BSM_PDR5']
     bsmpdr610 = ['BSM_PDR6', 'BSM_PDR7','BSM_PDR8', 'BSM_PDR9','BSM_PDR10']

     vanet_copy = vanet.copy()

     vanet_copy['BSM_PDR1-5_MEAN'] = vanet_copy[bsmpdr15].mean(axis=1)
     vanet_copy['BSM_PDR6-10_MEAN'] = vanet_copy[bsmpdr610].mean(axis=1)
     vanet_copy = vanet_copy.drop(['BSM_PDR1', 'BSM_PDR2','BSM_PDR3', 'BSM_PDR4','BSM_PDR5','BSM_PDR6', 'BSM_

     train2 = vanet_copy.iloc[0:np.rint(0.6*len(vanet_copy)).astype(int)]
     holdout2=vanet_copy.iloc[np.rint(0.6*len(vanet_copy)).astype(int):]

     cols_lr = select_features(train2,0)
     cols_la = select_features(train2,1)
     cols_ri = select_features(train2,2)
     cols_sgd = select_features(train2,3)


     print(cols_lr)
     print(cols_la)
     print(cols_ri)
     print(cols_sgd)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff3dd588c50>



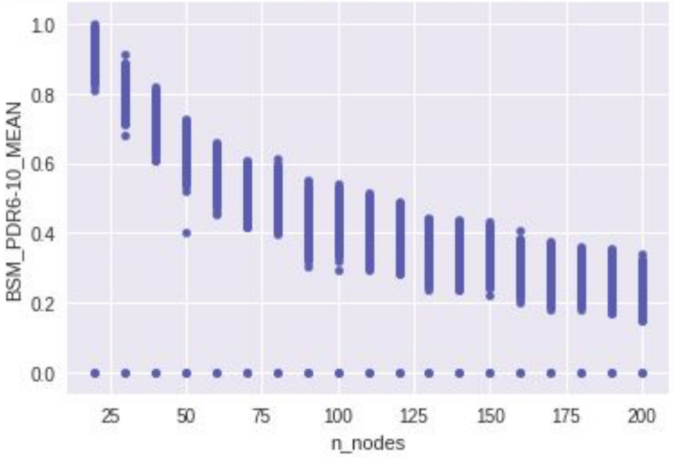
```
vanet_copy.plot(x='n_nodes', y='BSM_PDR1-5_MEAN', kind='scatter')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff3df016438>




```
vanet_copy.plot(x='n_nodes', y='BSM_PDR6-10_MEAN', kind='scatter')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff3df0117f0>



```python
clf_lr.fit(train2[cols_lr],train2["n_nodes"])
clf_la.fit(train2[cols_la],train2["n_nodes"])
clf_ri.fit(train2[cols_ri],train2["n_nodes"])
clf_sgd.fit(train2[cols_sgd],train2["n_nodes"])
clf_NNR.fit(train2[cols_lr],train2["n_nodes"])

pred_lr2 = clf_lr.predict(holdout2[cols_lr])
pred_la2 = clf_la.predict(holdout2[cols_la])
pred_ri2 = clf_ri.predict(holdout2[cols_ri])
pred_sgd2 = clf_sgd.predict(holdout2[cols_sgd])
pred_NNR2 = clf_NNR.predict(holdout2[cols_lr])

rmse_lr2 = np.sqrt(mean_squared_error(pred_lr2, holdout2["n_nodes"]))
rmse_la2 = np.sqrt(mean_squared_error(pred_la2, holdout2["n_nodes"]))
rmse_ri2 = np.sqrt(mean_squared_error(pred_ri2, holdout2["n_nodes"]))
rmse_sgd2 = np.sqrt(mean_squared_error(pred_sgd2, holdout2["n_nodes"]))
rmse_NNR2 = np.sqrt(mean_squared_error(pred_NNR2, holdout2["n_nodes"]))

print(rmse_lr2)
print(rmse_la2)
print(rmse_ri2)
print(rmse_sgd2)
print(rmse_NNR2)
```

```
/usr/local/lib/python
  FutureWarning)
3.060806106027885
3.4212834002349433
3.07165700538336
3.0769844854047395
2.092692811380139
```

Conclusion 3

THANKS!

Any questions?

You can find us at

andersonnunes@protonmail.com

danielro@ufrn.edu.br

iagodiogenes@gmail.com

martinsdecastro23@gmail.com