Medicon Engineering Themes
Volume 3 Issue 2 August 2022
Research Article

A Naive Bayes Multi-class Weighted Classifier of Internet Packet flows over a MPLS Network

Roland Déguénonvo[1]*, Audace A.V. Dossou-Olory[2] and Max Fréjus O. Sanya[3]

[1]*Institut de Formation et de Recherche en Informatique, Université d'Abomey-Calavi, Bénin*

[2]*Institut National de l'Eau, Université d'Abomey-Calavi, Bénin*

[3]*Ecole Polytechnique d'Abomey-Calavi, Université d'Abomey-calavi, Bénin*

***Corresponding Author:** Roland Déguénonvo, Institut de Formation et de Recherche en Informatique, BP 526, Université d'Abomey-Calavi Abomey-calavi, Littoral, Benin.

Received: July 04, 2022; **Published:** July 10, 2022

DOI: 10.55162/MCET.03.062

Abstract

Our simulation is based first, on a qualitative approach for the classification of flows and traffic, and next on an experimental approach for the management of data volume on the other hand. The adopted approaches allowed us to get an idea on a NBWM (Naive Bayes Weighted Multi-class) classifier capable to output differentiated service classes in MPLS (Multiple Protocol Label Switching) networks. The classifiers we compared to our benchmark model were thoroughly processed. The accuracy rate of the proposed NBWM (Naïve Bayes Weighted Multiclass) classifier is about 68.75%, which puts it ahead of the other models encountered.

Keywords: Simulation; flow and traffic classification; NBMW classifier; MPLS networks; accuracy rate

Abbreviations

ANN: Artificial Neural Network.
BWNB: Binary Weighted Naïve Bayesian.
IP: Internet Protocol.
KNN: K Nearest Neighbor.
LER: Label Edge Router.
LR: Logistic Regression.
LSR: Label Switch Router.
MPLS: Multiple Protocol Label Switching.
NB: Naïve Bayes.
NBWM: Naïve Bayes Weighted Multiclass.
RF: Random Forest.
SVM: Support Vector Machine.
TCP: Transmission Control Protocol.

Introduction

In computer and telecommunication networks, the simulation of design work is one of the activities that is carried out using tools that are seen from several aspects. Simulation techniques have been used since the first communication networks were designed, but

until recently this practice has never been used as it should have been. Today simulation is making a comeback in network research activities on the Internet. Numerous simulation research works have been conducted around the world. In the context of bringing a better quality of service in Internet networks, we have adopted an experimental approach. We propose a Naive Bayes Multi-class Weighted (NBMW) classifier of Internet packet flows over a Multiple Protocol Label Switching (MPLS) network. We have adapted the classifier to a router that is able to route packets from the network input to the output without ambiguity. We will present the results of the simulations of the NBMW classifier. Indeed, these results will allow us to present the performance of the packet flow classifier on the one hand, and to compare it with other multiclass classifiers on the other hand.

Material and Methods

This work consists in simulating an NBMW classifier adapted to an MPLS router that is located at the edge of the MPLS network. The goal is to verify if the NBPM classifier can output more than two classes of services without losing packets on the network. The NBPM classifier will sort the packets and deduce the ones that require a better quality of service. Then we will compare this classifier to other reference models.

Material used

- Layer 3 router located at the edge of the network.
- Computer with an Anaconda environment 2019.03 version 3.7.7 of the Python Language.

Methods

First, we have chosen Binary Weighted Naïve Bayesian (BWNB) classifier limited by two data outputs as indicated on Figure 1:

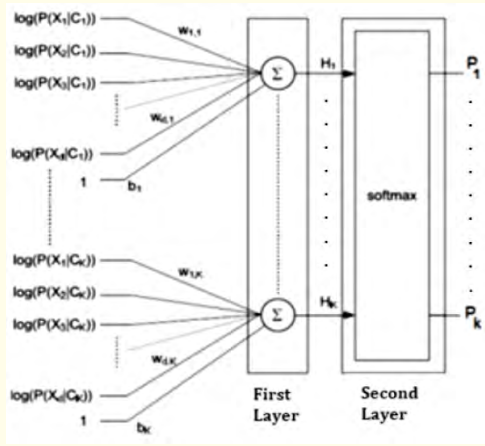


Figure 1: Graphical model for weight optimization of the averaged Bayesian classifier.

The BWNB classifier on Figure 1 is characterized by two layers of which the equations are:

$$P(C_k / X) = \frac{P(C_k)\Pi_i P(X_i \mid C_k)^{\omega i}}{\sum_{j=1}^{K}(P(C_j)\Pi_i P(X_i \mid C_j)^{\omega i})} \quad (1)$$

Where each explanatory variable i is weighted by a weight ω_i in the interval [0, 1], j is the class index ($j \in \{1,\ldots K\}$), and k a class of interest.

The first layer of the graphical model (Figure 1) is a linear layer performing a sum H_k for each class k, such that:

$$H_k = \sum_{i=1}^{d} w_{ik} \log(P(X_i|C_k) + b_k. \quad (2)$$

The second layer of the graphic model is a Softmax such that:

$$P_k = \frac{e^{H_k}}{\sum_{j=1}^{K} e^{H_j}} \quad (3)$$

Where:

- w_{ik} is the weight associated with variable i and class k ,
- b_k is the bias associated with class k

We have modeled the BWNB equations and obtained the NBMW that we have proposed. We have then adapted the new proposed approach (NBMW) at border router in MPLS Network as indicated in Figure 2.

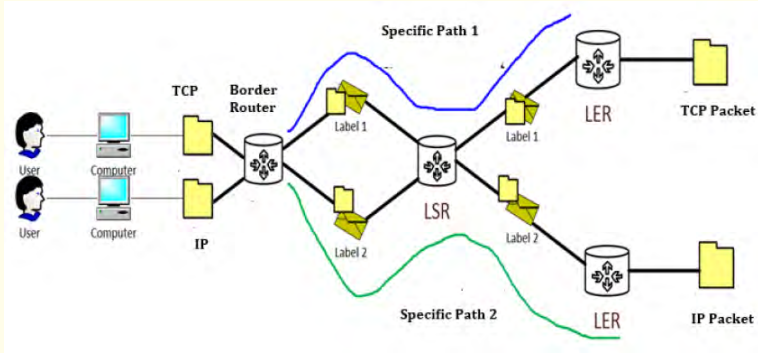


Figure 2: Routing of packet flows in the MPLS Network.

The proposed NBMW approach is a multiclass classifier (Cf. Figure 3) and characterized by the following equations:

$$\frac{\partial F}{\partial z_k} = -A_k + P_k \sum_{j=1}^{t} A_j \quad (4)$$

$$\frac{\partial F}{\partial z_k} = P_k - A_k \quad (5)$$

A neural network generates (based on the available learning data) the values $z_1(X)$,$_2(X)$,.....,$z_t(X)$. These values are then injected into a Softmax, which is nothing but a multinomial logistic regression function. More generally, Softmax $(x_j) = \frac{e^{x_j}}{\sum_{i=1}^{t} e^{x_i}}$. This allows us to obtain the following diagram:

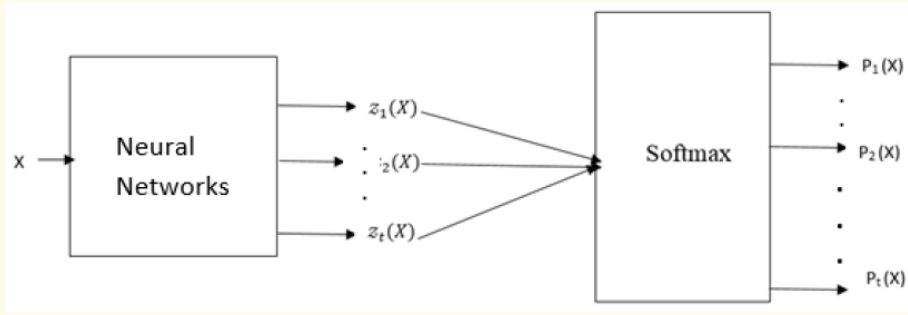


Figure 3: Illustration of the multiclass approach.

In Figure 3, the neural network takes as input the x-values, computes the sum of the x-values, and then these values pass through the activation function to produce the outputs $z_1(X)$, $z_2(X)$,....., $z_t(X)$. Then the outings $z_1(X)$, $z_2(X)$,....., $z_t(X)$ are injected into a Softmax. The Softmax takes as inputs the values $z_1(X)$, $z_2(X)$,....., $z_t(X)$ and generates the output vectors P(X) having t positive real-valued components whose sum is 1. The Softmax function is defined by:

$$P_j(X) = \frac{e^{z_j(X)}}{\sum_{i=1}^{t} e^{z_i(X)}}, \forall j \in \{1,2,...,t\} \qquad (6)$$

We then compare $P_1(X)$,...,$P_t(X)$ with the different thresholds associated with each class, and we conclude to which class X belongs.

Simulations

To perform the simulations, we used the Anaconda environment version 2019.03 with the Python language version 3.7.7. Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing. It aims to simplify package management and deployment. It is a distribution that includes data science packages suitable for Linux, MacOs and Windows. The classification of packet flows is done according to measured characteristics. The characteristics include class of service, throughput, latency, transmission time and amount of data exchanged on each flow. All these characteristics were measured on a real network.

Training of the classifier

We have the following training data set from a real network analyzed with Wireshark:

Class of service	Throughput	Latency	Amount of data exchanged on each flow	Protocol	Port	Duration	Technology used
Messaging	1.4kbps	192.2ms	544.4kb	DNS	N/A	47min	N/A
WEB	30.7kbps	5ms	12MB	HTTP	N/A	52min	N/A
Network	5.4Mbps	66.2ms	2.1Gb	TCP	N/A	52min	N/A
WEB	54bps	151.8ms	21.1kb	N/A	N/A	1min	Mozilla
WEB	8.7bps	0	3.4kb	N/A	N/A	0min	CIFS (Common Internet File System)
WEB, VIDEO	9.4kbps	315.5	3.7MB	N/A	N/A	60min	YouTube
Network	38.7bps	0	15.1kb	NTP	N/A	0min	N/A
WEB	6.8kbps	191.5	2.7MB	N/A	N/A	53 min	Google (WS, online search)
VIDEO	2.2kbps	203.8	843.6kb	N/A	N/A	50min	Skype
Filtering	273.6kbps	183	106.9MB	SSL	N/A	52min	N/A
WEB	1.2kbps	141.3	470.1kb	N/A	N/A	50min	AddThis (user analytics)
WEB	702.7bps	149.2	274.5kb	N/A	N/A	50min	Google API
Filtering	105.6bps	154.9	41.2kb	N/A	N/A	65min	OCSP (key verifier)
WEB	82.3bps	149.6	32.2kb	N/A	N/A	30min	BING
WEB	959.5bps	120.9	374.8kb	N/A	N/A	40min	Facebook
WEB	119.9bps	146	46.8kb	N/A	N/A	1min	Google Analytics
WEB	39.9bps	141ms	15.6kb	N/A	N/A	1min	LinkedIn
WEB	152.1bps	230.4ms	59.4kb	N/A	N/A	70min	DoubleClick
WEB	1.8 kbps	244.1ms	714.7kb	N/A	N/A	60min	Gmail
Filtering	2.3kbps	143.9ms	882.7kb	SSH	N/A	30min	N/A
Storage	2.3kbps	143.9ms	882.7kb	SFTP	N/A	30min	N/A
N/A	59.3bps	0	23.2kb	SSDP	N/A	1min	N/A
N/A	1.8bps	0	690B	MDNS	N/A	N/A	N/A
WEB	144.3bps	206.5ms	56.3kb	N/A	N/A	1min	AWS
WEB	22.4bps	306.6ms	8.7kb	N/A	N/A	1min	Google Safe br
WEB	4.3Mbps	196.1	1.7Gb	N/A	N/A	53min	Microsoft
Messaging	1.3kbps	204.5	511kb	N/A	N/A	53min	MS Office 365
NetBIOS Name Service	92 octets	0.0668610 00 seconds	736bits	NBNS	N/A	128	Wireshark
WEB	6.9bps	147.4	2.7kb	N/A	N/A	1min	MSN
N/A	64 octets	0.2262080 00 seconds	512bits	N/A	N/A	N/A	Wireshark
N/A	20.8bps	118.9	8.1kb	N/A	N/A	1 min	BITS
N/A	89 octets	0.0000030 00 seconds	712bits	LLMNR	N/A	N/A	Wireshark
N/A	21.4bps	0	8.4kb	N/A	N/A	1min	STUN
Voice	60 octets	158869258 8.6962380 00 seconds	480bits	Udp	N/A	255	Wireshark
Web	194 octets	0.0006120 00 seconds	1552bits	http	80	128	Wireshark
DHCP	342 octets	0.0032790 00 seconds	2736bits	N/A	N/A	N/A	Wireshark
WEB	13.7kbps	149.8ms	5.4Mb	N/A	N/A	30mn	Google Drive
DNS	208 octets	0.2327800 00 seconds	1664bits	Udp	N/A	102	Wireshark
WEB	1.2kbps	185.9	481.7kb	N/A	N/A	70min	Windows live
WEB	689.8 kbps	176ms	269.4MB	N/A	N/A	52min	Windows Update
WEB	207bps	145.1ms	80.8kb	N/A	N/A	1min	Google Play
WEB	26.2bps	147.6ms	10.2kb	N/A	N/A	1min	Twitter

Table 1: Packet flow data of a real network.

Data processing

We processed the data in the following steps:

Import of Libraries

```
In [1]: import matplotlib, sklearn, sys
        import matplotlib.pyplot as plt
        import tensorflow as tf
        import numpy as np
        import pandas as pd
        from sklearn.model_selection import cross_val_score , validation_curve
        from sklearn.preprocessing import LabelEncoder, OrdinalEncoder
```

Library version

```
In [2]: print("Version Tensorflow: ",tf.__version__)
        print("Version Matplotlib: ",matplotlib.__version__)
        print("Version Numpy: ",np.__version__)
        print("Version Pandas: ",pd.__version__)
        print("Version Scikit learn: ",sklearn.__version__)

        print("Version Python: ",sys.version)

        Version Tensorflow:  2.1.0
        Version Matplotlib:  3.1.3
        Version Numpy:  1.18.1
        Version Pandas:  1.0.3
        Version Scikit learn:  0.22.1
        Version Python:  3.7.7 (default, Mar 26 2020, 15:48:22)
        [GCC 7.3.0]
```

Data Import

```
In [3]: data = pd.read_excel("base2.xlsx")
        data.columns = ['Classe', 'Debit', 'Latence', 'Data_Xchange', 'Duree']
        print(data.dtypes)
        data

        Class            object
        Throughput      float64
        Latency         float64
        Data Xchange    float64
        Duration          int64
        dtype: object
```

Figure 4: Data processing steps.
(*Source:* Tensor Flow file generated by Anaconda simulator).

Preprocessing

```
In [4]: df = data.copy()
        target = df.loc[:,"Classe"]
        features = df.loc[:, "Debit":"Duree"]

        from sklearn.preprocessing import LabelBinarizer, RobustScale.
        #encoder = LabelBinarizer(sparse_output= False)
        encoder = LabelEncoder()
        target_enc = encoder.fit_transform(data.loc[:,'Classe'])
        print(target_enc.shape)
        target_enc = np.reshape(target_enc,(target_enc.shape[0]))
        stand = RobustScaler()
        features_trans = stand.fit_transform(features)

        (32,)

In [5]: target_enc

Out[5]: array([1, 4, 2, 4, 4, 3, 2, 4, 3, 0, 4, 4, 0, 4, 4, 4, 4, 4, 4, 0, 0, 4,
               4, 4, 1, 4, 4, 4, 4, 4, 4, 4])
```

Figure 5: Preprocessing of the measured parameters.
(*Source:* Tensor Flow file generated by the Anaconda simulator).

Out[3]:

	Class	Throughput	Latency	sta_Xchangı	Duration
0	Messaging	1.4000	192.200	5.444000e+02	47
1	WEB	30.7000	5.000	1.228800e+04	52
2	Network	5400.0000	66.200	2.202010e+06	52
3	WEB	0.0540	151.800	2.110000e+01	1
4	WEB	0.0087	0.000	3.400000e+00	0
5	VIDEO	9.4000	315.500	3.788800e+03	60
6	Network	0.0387	0.000	1.510000e+01	0
7	WEB	6.8000	191.500	2.764800e+03	53
8	VIDEO	2.2000	203.800	8.436000e+02	50
9	Filtering	73.6000	183.000	1.094656e+05	52
10	WEB	1.2000	141.300	4.701000e+02	50
11	WEB	0.7027	149.200	2.745000e+02	50
12	Filtering	0.1056	154.900	4.120000e+01	65
13	WEB	0.0823	149.600	3.220000e+01	30
14	WEB	0.9595	120.900	3.748000e+02	40
15	WEB	0.1199	146.000	4.680000e+01	1
16	WEB	0.0399	141.000	1.560000e+01	1
17	WEB	0.1521	230.400	5.940000e+01	70
18	WEB	1.8000	244.100	7.147000e+02	60
19	Filtering	2.3000	143.900	8.827000e+02	30
20	Filtering	2.3000	143.900	8.827000e+02	30
21	WEB	0.1443	206.500	5.630000e+01	1
22	WEB	0.0224	306.600	8.700000e+00	1
23	WEB	4300.0000	196.100	1.782579e+06	53
24	Messaging	1.3000	204.500	5.110000e+02	53
25	WEB	0.0069	147.400	2.700000e+00	1
26	WEB	1.5520	0.612	1.515625e+00	128
27	WEB	13.7000	149.800	5.529600e+03	30
28	WEB	1.2000	185.900	4.817000e+02	70
29	WEB	689.8000	176.000	2.758656e+05	52
30	WEB	0.2070	145.100	8.080000e+01	1
31	WEB	0.0262	147.600	1.020000e+01	1

Table 2: Packet flow data obtained after processing.

Model

```
In [6]: from tensorflow import keras

        initial_learning_rate = 0.001
        lr_schedule = keras.optimizers.schedules.ExponentialDecay(
            initial_learning_rate,
            decay_steps=100000,
            decay_rate=0.96,
            staircase=True)


        def create_model(input_dim):#adam
            model = tf.keras.models.Sequential()
            model.add(tf.keras.layers.Dense(4, input_dim=input_dim, activation="relu"))

            model.add(tf.keras.layers.Dense(5, activation="sigmoid"))

            model.add(tf.keras.layers.Dense((target_enc.max())+1, activation="softmax"))#sparse_categorical_crossentropy_with_logits

            model.compile(
                loss= keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                optimizer=keras.optimizers.Adam(learning_rate=lr_schedule),#
                metrics=["accuracy"]

            )

            return model
```

Figure 6: Training of the proposed multiclass model.
(*Source:* TensorFlow file generated by the Anaconda simulator).

Results and Discussion

For the simulations, a multi-class Naive Bayes Weighted (NBP) classifier is used. It uses the estimates of conditional densities at the classes from the methods described in the previous sections. The results are presented in Figures 7-8 and show that the estimation of conditional probabilities is accurate for all training sets due to the fact that the multi-class Weighted Naive Bayes classifier performs well. The evaluation of our model gives an accuracy of 0.6875.

Figure 8 (b) refers to the cost of our classifier that is minimized. The blue curve shows the evolution of the minimization of the cost of our classifier according to each iteration performed on the training data. On the other hand, the orange curve refers to the evolution of the minimization on the validation data. We can see that at each iteration, the cost of the model decreases which effectively confirms the minimization.

Figure 8 (c) gives the values of the accuracy of our model as a function of the number of iterations. The two orange and blue curves in this figure 8 (c) illustrate the results obtained at each iteration on the training and validation data. A KNN classifier, which we trained with our simulation software gave an accuracy in predictions of 0.6875 (Cf. Figure 7). As others results:

- A Random Forest (RF) classifier that we trained with our simulation software gave an accuracy of 0.65625 (Cf. Figure 10).
- A Support Vector Machine classifier that we trained with our simulation software gave an accuracy of 0.5625 (Cf. Figure 11).
- An SGD classifier that we trained with our simulation software gave an accuracy of 0.53125 (Cf. Figure 12).

Estimation with Stratified fold

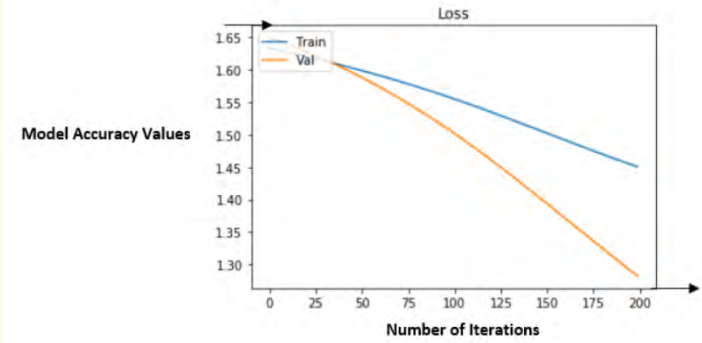
```
In [7]: from sklearn.model_selection import StratifiedKFold
        for train_index,test_index, in StratifiedKFold(2).split(features_trans, target_enc):
            features_train, features_test = features_trans[train_index], features_trans[test_index]
            target_train, target_test = target_enc[train_index], target_enc[test_index]

            model = create_model(features_trans.shape[1])
            history = model.fit(features_train, target_train, epochs=200, validation_split=0.2)

            print('Model evaluation ', model.evaluate(features_test, target_test))

        Train on 12 samples, validate on 4 samples
```

Figure 7: KNN model performance evaluation and its implementation.

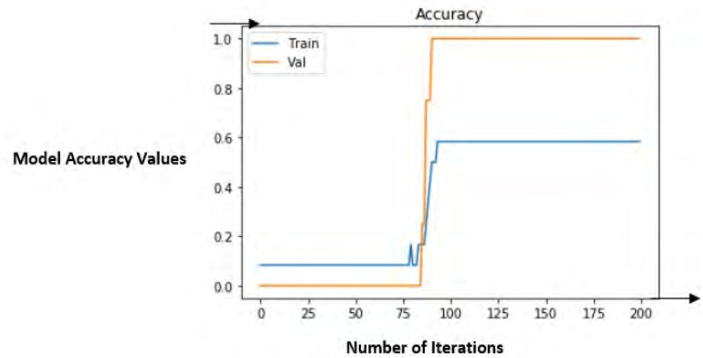
```
In [8]: loss_curve = history.history["loss"]
        acc_curve = history.history["accuracy"]

        loss_val_curve = history.history["val_loss"]
        acc_val_curve = history.history["val_accuracy"]

        plt.plot(loss_curve, label="Train")
        plt.plot(loss_val_curve, label="Val")
        plt.legend(loc="upper left")
        plt.title("Loss")
        plt.show()

        plt.plot(acc_curve, label="Train")
        plt.plot(acc_val_curve, label="Val")
        plt.legend(loc="upper left")
        plt.title("Accuracy")
        plt.show()
```

```
16/16 [==============================] - 0s 1ms/sample - loss: 1.4196 - accuracy: 0.6875
Model evaluation  [1.4195828437805176, 0.6875]
```

(a)



(b)



(c)

Figure 8: Model validation curves-(a) simulation results in terms of loss and accuracy-(b) Classifier cost minimization versus Number of iterations-(c) Classifier Accuracy values versus Number of iterations.

KNN

```
In [9]: from sklearn.neighbors import KNeighborsClassifier
        from sklearn.model_selection import GridSearchCV
        model = KNeighborsClassifier()
        param_grid = {
            'n_neighbors': np.arange(1, 6),
            'metric': ['euclidean', 'manhattan']
        }

        n_split = 2
        grid = GridSearchCV(model,param_grid,cv=n_split)
        grid.fit(features_trans, target_enc)
        print(grid.best_score_)
        print(grid.best_params_)

        0.6875
        {'metric': 'euclidean', 'n_neighbors': 5}
```

Figure 9: Accuracy rate of the KNN classifier and its implementation.

Random Forest

```
In [10]: from sklearn.ensemble import RandomForestClassifier
         model2 = RandomForestClassifier()

         param_grid2 = {
             'max_depth': np.arange(1,6),
             'random_state': [0,1,2,3,4,5]

         }

         grid2 = GridSearchCV(model2, param_grid2,cv=n_split)
         grid2.fit(features_trans, target_enc)
         print(grid2.best_score_)
         print(grid2.best_params_)

         0.65625
         {'max_depth': 1, 'random_state': 0}
```

Figure 10: Accuracy rate of the Random Forest classifier and its implementation.

Support Vector Machine

```
In [11]: from sklearn.svm import LinearSVC
         model3 = LinearSVC()
         param_grid3 = {
             'random_state': [np.random.randint(1,11),'None']
         }
         grid3 = GridSearchCV(model3, param_grid3, cv=2)
         grid3.fit(features_trans, target_enc)
         print(grid3.best_score_)
         print(grid3.best_params_)

         0.5625
         {'random_state': 10}
```

Figure 11: Accuracy rate of the Support Vector Machine classifier and its implementation.

SGD Classifier

```
In [12]: from sklearn.linear_model import SGDClassifier
         model4 = SGDClassifier()
         param_grid4 = {
             "eta0" : np.linspace(0.1, 3.0, num=10),
             "learning_rate" : ["constant", "invscaling", "optimal", "adaptive"]
         }

         grid4 = GridSearchCV(model4, param_grid4,cv=n_split)
         grid4.fit(features_trans, target_enc)
         print(grid4.best_score_)
         print(grid4.best_params_)

         0.53125
         {'eta0': 0.1, 'learning_rate': 'invscaling'}
```

Figure 12: Accuracy rate of the SGD Classifier and its implementation.

Accuracy Rate	***NBMW***	***KNN***	***Random Forest***	***Support Vector Machine***	***SGD classifier***
	0.6875	0.6875	0.65625	0.5625	0.53125

Table 3: Performance comparison of simulated multiclass classifiers according to the accuracy rates.

The comparison of accuracy rate of our classifier NBWM with other references model (Cf. Table 3) shows that NBMWN and KNN have the same accuracy (0.6875).

Criteria	***NB***	***LR***	***SVM***	***RF***	***KNN***	***ANN***
Error rate	1	4	5	5	2	5
Exploitation of mixed type characteristics	1	3	3	5	3	3
Incorporation of information from C	2	4	3	5	1	3
Selection of a minimum of hyper parameters	5	4	1	5	1	2
Problematic optimization	5	4	1	5	5	1
Rating/5	2.8	3.6	2.6	5	3.2	2.8

Table 4: Evaluation of the classifiers according to the determining criteria.
(The scores are relative, ranging from 1 to 5).

We implement the criteria of Table 4 to compare the error rate of our NBMW model with KNN. According to Figure 8, the training of our model resulted in forecast errors of 31.25% (0.3125) of the cases. So out of 100 predictions, about 31 might only be wrongly predicted compared of 69 that can be well predicted. The error rate of KNN is 2 while our model NBMW show the error rate equal to 0.3125.

Our work consists in bringing a better quality of service to MPLS networks. We have treated a state of the art to have a better visibility of the work already done in the field. The synthesis of the work carried out shows that there are aspects of our work that have not been addressed. This allowed us to better position ourselves in order to achieve our objectives. Indeed, we have proposed a mathematical equation of the model that can provide more than two classes in output. We have also simulated the mathematical model equation with Anaconda version 2019.03 and then proceeded to its validation. The performance of the multi-class Naive Bayes Weighted classifier with an accuracy of 0.6875 is an important information for packet flow classification in an MPLS network. The model is able to give reliable statistics on which flows are of good quality and which are not useful to continue their way on the network. Also, the more data that is provided at the input of the network and the more the model is trained, the better the results will be. The comparative results of multiclass classifiers with our model leave behind Random Forest, Support Vector Machine, SGD classifiers. Our model has the same

accuracy (0.6875) as the KNN classifier but the latter is sensitive to the variables which we did not notice in the Naive Bayes Weighted classifier. The error rate of KNN is equal to 2, which is less performed than 0.3125 of the proposed NBMW model. The KNN can produce a good error rate but its method does not build a model, so it gives precisions without taking them into account. Moreover, it is a good generalist classifier, non-parametric, not requiring optimization.

In the case of our research, the training of the model has shown that when the error rates are low, the classifier performs well. From these results, we can deduce that since NBMW has a lower error rate than KNN, it is the most efficient.

Conclusion

Simulating a multiclass classifier in an MPLS network ingress router is a major challenge for us. The best results we have obtained from our classifier will provide good accuracy on classification activities. Accurate classification in the network will ensure good QoS of the packet flows. The comparison of our classifier with several reference models allowed an unambiguous validation of the model. With the increase in data flows on the Internet, modeling classifiers remains an ongoing challenge to further minimize conditional estimates. With the relentless development of the Internet, the world should expect more advanced research in the processing of large volume of data.

References

1. Christian Jacquenet. "Background MPLS Broadcast - The Best of IP Multicast and Traffic Engineering". TEA Engineering Techniques MPLS Broadcast (2013).
2. Salperwyck., et al. "Weighted Bayes naïve classifier for data flow". EGC 26 (2014): 275-286.
3. Aballo OT., et al. "Metrology in MPLS Network. Excellence in Research and Innovation (EIRAI) (2018): 21-23.
4. OP Owezarski Philippe. "Metrology of Internet networks: main actions and impact on technological developments". What metrology tells us about the future of the Internet (2003).
5. Luckie M., et al. "Network Hygiene, Incentives, and Regulation: Deployment of Source Address Validation in the Internet" In Proceedings of 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19) New York, NY, USA: ACM Press" (2019): 465-480.
6. K Jeremy., et al. "Hypersparse Neural Network Analysis of Large-Scale Internet Traffic". IEEE Conference Publishing (2019).
7. C Salperwyck., et al. "A two layers incremental discretization based on order statistics". Statistical Models for Data Analysis (2013): 315-323.
8. Hoeting., et al. "Correction to: Bayesian model averaging: a tutorial'' [Statist. Sci. 14 (1999), no. 4, 382--417; MR 2001a:62033]". Statist Science 15.3 (2000): 193-195.
9. Salperwyck. "Incremental online learning on data flow". Political Science, Computer Science. Online incremental learning on data streams - TEL - Theses online (2013).
10. Faten Mohamed Zhani. "Internet Traffic Forecasting - Models and Applications". Thesis: Montreal (Quebec, Canada), University of Quebec in Montreal, Doctorate in Computer Science (2011).
11. V Vapnik. "The nature of statistical learning theory". 2nd Edition, Springer, Berlin (2000).
12. Nicolay X., et al. Metrology of IP networks (2016).
13. Fs Rode rick., et al. "Unintended consequences: Effects of submarine cable deployment on Internet routing". International Conference on Passive and Active Network Measurement. PAM (2020): 211-227.
14. Rahul Kumar Sharma and V Sukumaran. "A comparative study of naive Bayes classifier and Bayes net classifier for fault diagnosis of roller bearing using sound signal" Int. J. Decision Support Systems 1.1 (2015).
15. Sona Taheri., et al. "Attribute weighted Naive Bayes classifier using a local optimization". Springer (2013).
16. Andrew Y Ng and Michael I Jordan. "On Discriminative vs. Generative classifiers: A comparison of Logistic regression and naive Bayes" (2001).

17. Alexander Genkin., et al. "Large-Scale Bayesian Logistic Regression for Text Categorization". Technometrics 49.3 (2007): 291-304.
18. Fereshteh Falah Chamasemani., et al. "Multi-class Support Vector Machine (SVM) Classifiers-An Application in Hypothyroid Detection and Classification" 2011 Sixth International Conference on Bio-Inspired Computing: Theories and Applications (2011).
19. Jing Wang., et al. "Risk assessment of coronary heart disease based on cloud-random forest". Springer (2022).
20. Lei Xiong and Ye Yao. "Study on an adaptive thermal comfort model with K-nearest-neighbors (KNN) Algorithm". Elsevier, Building and Environment, Science Direct 202 (2021): 108026.
21. Oludare Isaac Abiodun., et al. "State-of-the -art in artificial neural network applications: A survery". Science Direct Heliyon 4.11 (2018): e00938.

Volume 3 Issue 2 August 2022