CS5051.500 Machine Learning Course Assignment 2 Final Report

Team:
A01271904 Laura Jaideny Pérez Gómez
A01271549 Juan Carlos Ángeles Cerón
A00829703 Andreé Vela Miam

Assignment description

The following report describes the results obtained from the assignment of the Computational Techniques of Machine Learning course, based on the work of Medina et al. [1], which describes the implementation of the Validity Index using supervised Classifiers (VIC). This new Index uses supervised classifiers, which will work as artificial experts making a decision. Using different classifiers allows us to mitigate the bias associated with each classifier and obtain a partition quality, which will be better as the classification performance increases.

From a partition in a given dataset, VIC obtains each element's labels within the dataset to build and train a set of supervised classifiers. In this way, assuming that the partition has good quality, each classifier should correctly predict the class of any observation. For the assignment's development, we found 50 different partitions for two and three clusters of the fingerprint dataset, which contains numeric attributes related to minutiae fingerprints. The main objective is to evaluate each partition using VIC with a set of 7 different supervised classifiers to find the best

Methodology

Handling missing values

The dataset used contains several missing values. Since some of the classification algorithms do not know how to deal with these values, it was necessary to carry out data imputation to estimate the missing values. For this purpose, an analysis was carried out to determine the percentage of missing values, with which we found a total of 42 columns with missing values. A first approach to handling the missing values was using a multivariate iterative imputation. This method offers an estimation closer to the real value than other techniques (e.g., fill forward, fill backward, fill mean). This technique's essential operation is as follows: for each column c_i with empty values, an estimator is trained using all the other columns as features, and c_i as labels. For training, the instances that contain values in c_i are used, and the model is finally used to estimate the empty instances of c_i.

However, after some experimentation, it was concluded that this technique results in a significant increase in the training time (more than 6 hours per partition). Hence, this technique was discarded in favor of a simpler method. That method was filling the missing values using the mean of the attribute. Finally, the total execution time for the 50 partitions was about two hours.

Finding clusters

Two-cluster partitions

The value of the score_change attribute within the dataset was sorted from smallest to largest to generate the partitions. For the two-cluster partitions, a partition was generated at each end with 30 elements each, to ensure that no partition with less than 30 elements exists. The remaining set was then divided into 50 parts, based on the score_change by dividing the complete range of values into 50 uniformly spaced samples. These samples were calculated on a start-end interval, starting from the element following the end of the first partition to the element before the beginning of the last partition. In the end, 50 different partitions with two classes labeled as '+' and '-' were obtained. Figure 1 shows a visual representation of how the two classes, represented as orange and green, were generated in each partition. Table 1 presents the detailed information of the partitions.

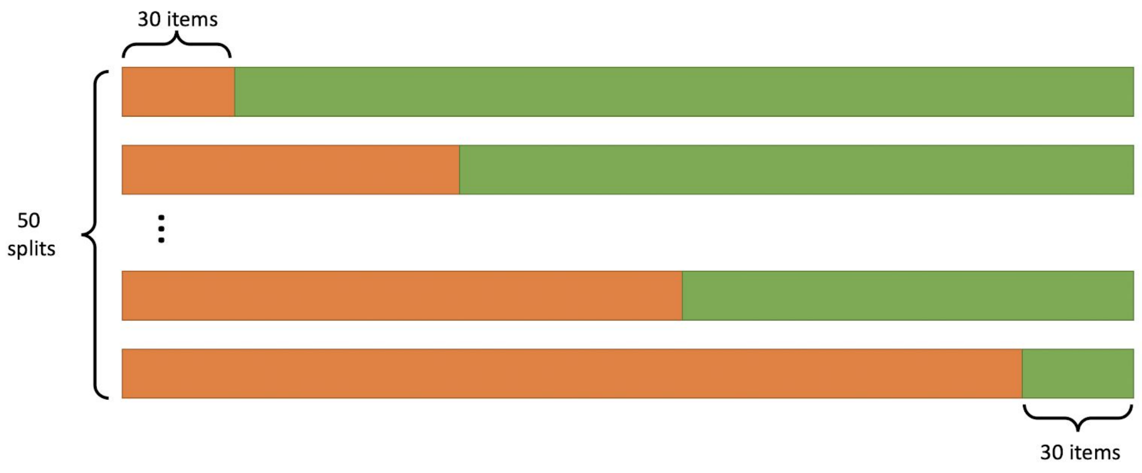

Figure 1. Procedure to generate the two-cluster partitions.

Table 1. Description of the two-cluster partitions.

partition	split	c1_size	c2_size	partition	split	c1_size	c2_size
1	-0.8976	31	5210	26	-0.1568	1472	3769
2	-0.8679	35	5206	27	-0.1271	1758	3483
3	-0.8383	41	5200	28	-0.0975	2082	3159
4	-0.8087	47	5194	29	-0.0679	2407	2834
5	-0.779	56	5185	30	-0.0383	2752	2489

6	-0.7494	64	5177	31	-0.0086	3115	2126
7	-0.7198	76	5165	32	0.021	3558	1683
8	-0.6901	91	5150	33	0.0506	3935	1306
9	-0.6605	107	5134	34	0.0803	4208	1033
10	-0.6309	128	5113	35	0.1099	4403	838
11	-0.6013	141	5100	36	0.1395	4609	632
12	-0.5716	157	5084	37	0.1692	4743	498
13	-0.542	176	5065	38	0.1988	4852	389
14	-0.5124	206	5035	39	0.2284	4920	321
15	-0.4827	233	5008	40	0.2581	4982	259
16	-0.4531	275	4966	41	0.2877	5042	199
17	-0.4235	322	4919	42	0.3173	5080	161
18	-0.3938	365	4876	43	0.347	5111	130
19	-0.3642	437	4804	44	0.3766	5140	101
20	-0.3346	523	4718	45	0.4062	5162	79
21	-0.3049	623	4618	46	0.4359	5172	69
22	-0.2753	750	4491	47	0.4655	5186	55
23	-0.2457	892	4349	48	0.4951	5200	41
24	-0.216	1059	4182	49	0.5247	5206	35
25	-0.1864	1252	3989	50	0.5544	5211	30

Three-cluster partitions

A process similar to that described for the case of two clusters was followed to generate the partitions of three sets. First, the observations were ordered according to the score_change attribute. Later, two partitions were generated at each end with 30 elements each, along with a new central partition obtained from the database's median. This last partition was reduced in each split until it remained with 30 elements, considering the 15 elements before and after the median. The remaining two sets were divided into 50 parts, the first of these sets found after the first partition until the median partition, and the second one from the end of the median partition until the beginning of the last partition. Each partition was generated based on the score_change value obtained from 50 samples evenly spaced on a start-end interval. In the end, 50 different partitions with three classes labeled as '+', '0', and '-' were obtained. Figure 2 shows a visual representation of how the three classes, represented as orange, blue, and green, were generated in each partition. Table 2 presents the detailed information of the partitions.

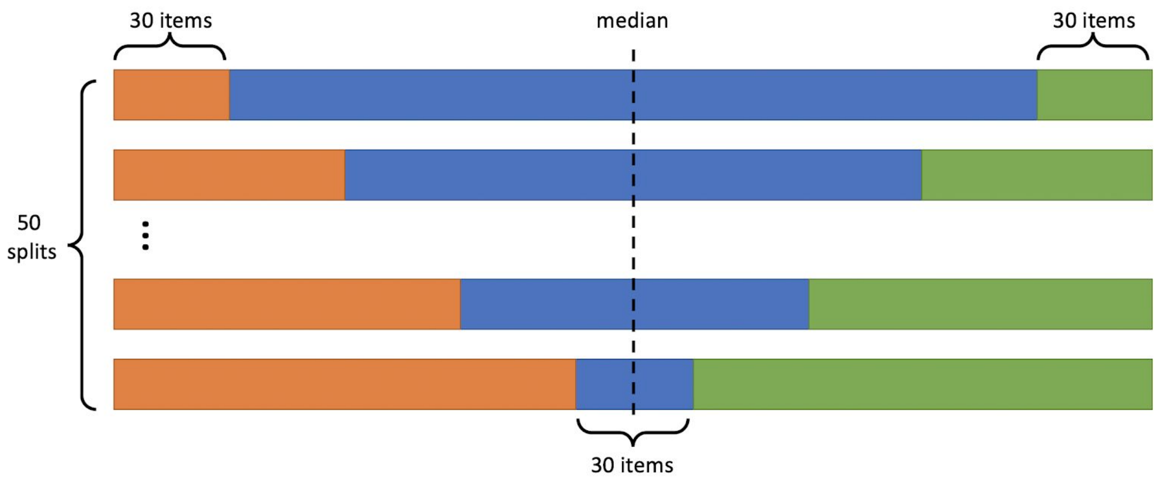

Figure 2. Procedure to generate the three-cluster partitions.

Table 2. Description of the three-cluster partitions.

par	split 1	split 2	c1 size	c2 size	c3 size	par	split 1	split 2	c1 size	c2 size	c3 size
1	-0.8976	0.5544	31	5180	30	26	-0.4648	0.2472	260	4702	279
2	-0.8803	0.5421	31	5178	32	27	-0.4475	0.2349	282	4653	306
3	-0.8629	0.5298	36	5172	33	28	-0.4302	0.2226	309	4600	332
4	-0.8456	0.5175	40	5164	37	29	-0.4129	0.2103	331	4547	363
5	-0.8283	0.5052	42	5161	38	30	-0.3956	0.1981	362	4487	392
6	-0.811	0.4929	47	5153	41	31	-0.3783	0.1858	391	4421	429
7	-0.7937	0.4807	51	5145	45	32	-0.361	0.1735	451	4307	483
8	-0.7764	0.4684	56	5134	51	33	-0.3437	0.1612	496	4221	524
9	-0.7591	0.4561	62	5117	62	34	-0.3264	0.1489	560	4100	581
10	-0.7418	0.4438	66	5109	66	35	-0.309	0.1366	611	3981	649
11	-0.7245	0.4315	75	5095	71	36	-0.2917	0.1243	688	3827	726
12	-0.7072	0.4192	84	5081	76	37	-0.2744	0.1121	753	3672	816
13	-0.6899	0.4069	91	5071	79	38	-0.2571	0.0998	840	3498	903
14	-0.6725	0.3947	98	5061	82	39	-0.2398	0.0875	917	3343	981
15	-0.6552	0.3824	112	5032	97	40	-0.2225	0.0752	1022	3146	1073
16	-0.6379	0.3701	120	5015	106	41	-0.2052	0.0629	1127	2933	1181
17	-0.6206	0.3578	133	4992	116	42	-0.1879	0.0506	1245	2690	1306
18	-0.6033	0.3455	140	4970	131	43	-0.1706	0.0383	1373	2401	1467
19	-0.586	0.3332	149	4948	144	44	-0.1533	0.026	1505	2120	1616
20	-0.5687	0.3209	158	4928	155	45	-0.136	0.0138	1683	1791	1767
21	-0.5514	0.3086	168	4897	176	46	-0.1186	0.0015	1860	1411	1970
22	-0.5341	0.2964	184	4867	190	47	-0.1013	0.0108	2039	1052	2150
23	-0.5168	0.2841	203	4832	206	48	-0.084	0.0231	2225	701	2315
24	-0.4995	0.2718	220	4795	226	49	-0.0667	0.0354	2416	366	2459
25	-0.4821	0.2595	233	4751	257	50	-0.0494	0.0477	2607	30	2604

Partition validation

The VIC implementation found in [2] was used to validate each of the partitions. This implementation is coded in Python and uses 7 Classifiers:

- Multilayer Perceptron
- AdaBoost
- K-Nearest Neighbors
- Random Forest
- Decision Tree
- Naive Bayes
- Linear Discriminant Analysis (LDA)

The VIC implementation was slightly modified to work with the database related to this assignment, but the core structure was conserved.

Results and Discussion

Two-classes

The best partition

Figure 3 shows the highest VIC score obtained in each of the 50 partitions. It can be seen that the partition with the highest score was the 50, with a 0.838 AUC score using the Naive Bayes algorithm. On the other hand, the worst classification performance was obtained in the partition 33, with a 0.577 AUC score using the LDA algorithm. Likewise, it can be seen that from partition number 20, the AUC score obtained begins to decrease until it reaches the minimum value and subsequently presents an increase until reaching its maximum value in the last partition.

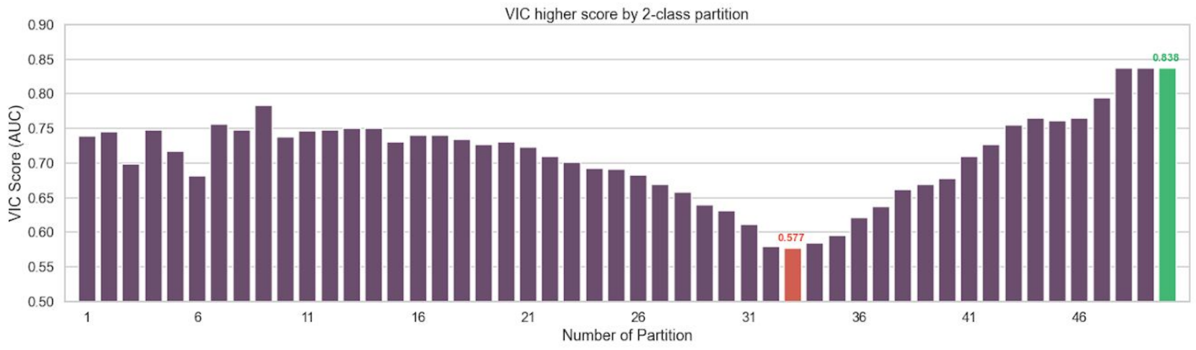

Figure 3. VIC higher score obtained for each partition.

Scores by algorithm

Figure 2 shows the AUC score obtained in each partition for each of the VIC classification algorithms. For each algorithm, it is shown which partition got the highest and the lowest AUC score. As can be seen, the performance of each partition using Naive Bayes was the one that was more similar to the overall VIC highest score graph. The worst performing classifier was K-NN, which obtained the lowest AUC score values compared to the rest of the classifiers, both in its best and worst partitions.

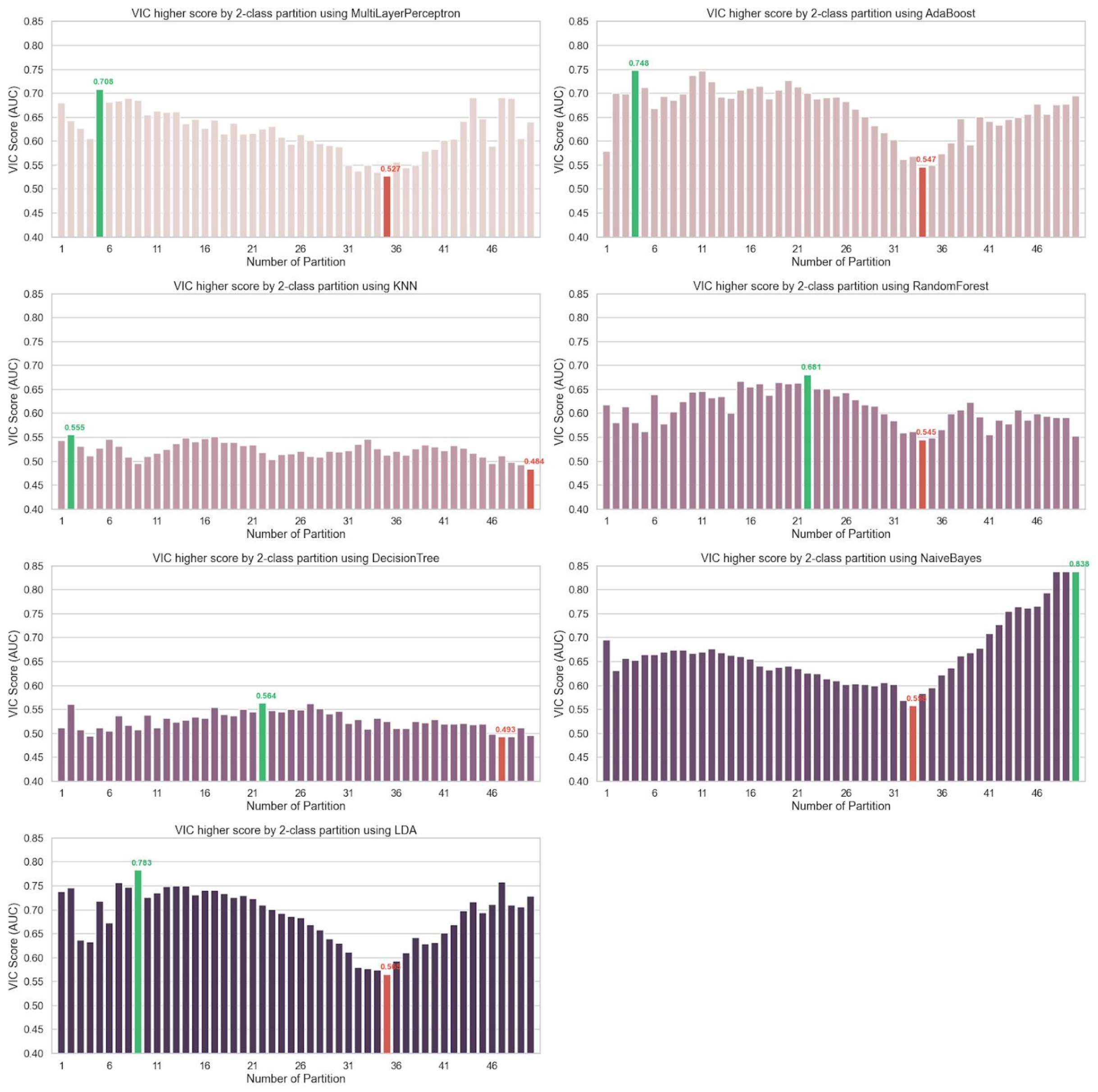

Figure 4. VIC higher score for each partition per classifier.

MVP

Figure 5 shows the number of times each algorithm obtained the highest AUC score in an individual partition. LDA and Naive Bayes were the ones that obtained the highest scores most frequently, with 27 and 17 partitions respectively. This result explains why the trend of both is similar to the overall VIC highest score graph. On the other hand, K-NN, Random Forest, and Decision Tree did not obtain the best AUC score in any partition. This poor performance could indicate that these classification algorithms are not suitable for the classification of this dataset. It could also suggest that the chosen hyper-parameters (i.e., sklearn default values) for those algorithms are not optimally tuned.

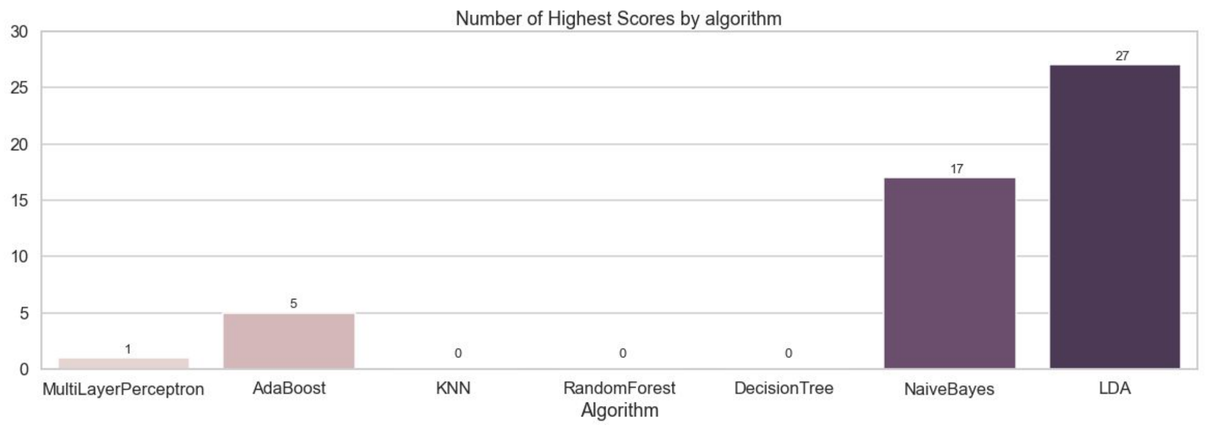

Figure 5. Number of VIC higher scores obtained per classifier.

Variance analysis by algorithm

Figure 6 shows a box plot that allowed us to analyze better the AUC score obtained on average by each algorithm. As expected, LDA was the algorithm that obtained the highest average AUC score. It was concluded that although Adaboost obtained an AUC mean higher than Naive Bayes, Naive Bayes obtained better results in more partitions due to the outlier values observed in the graph for this algorithm.

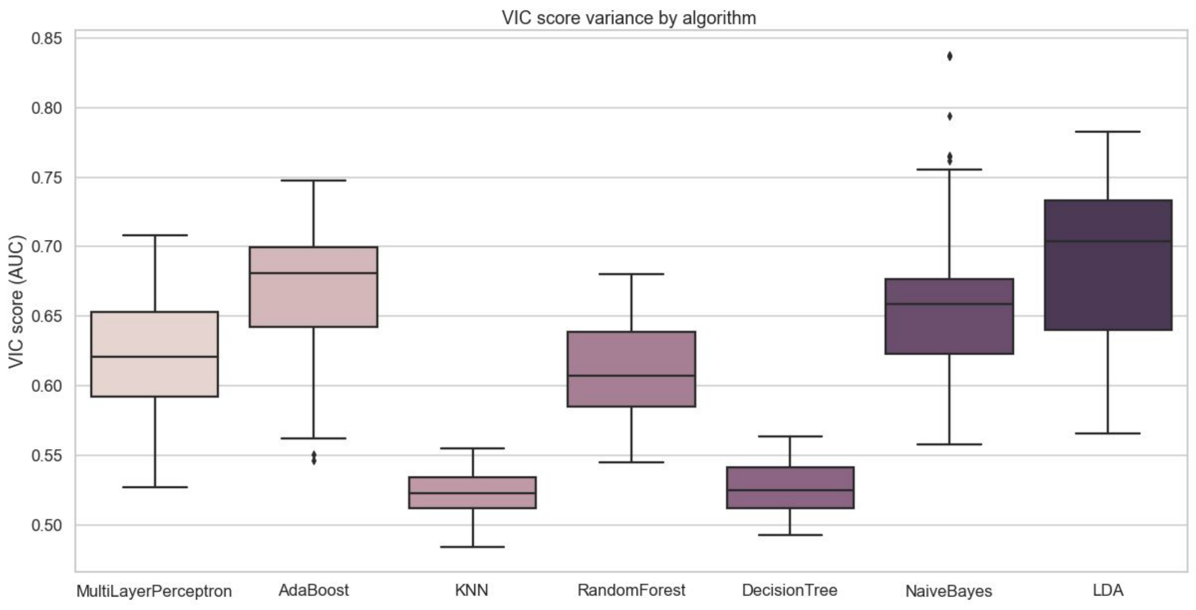

Figure 6. VIC score variance per classifier.

Three-classes

The best partition

Figure 7 shows the highest VIC score obtained in each of the 50 partitions. It can be seen that the partition 15 was the one that obtained the highest score, with a 0.746 AUC score using the Naive Bayes algorithm. The partition 50, on the other hand, obtained the worst classification performance, with 0.597 using the Random Forest algorithm. As can be seen, the behavior was quite different from the one obtained with two classes. For the tree-clusters case, the lowest AUC scores were obtained in the last partitions, while the highest AUC scores were maintained in the first partitions.

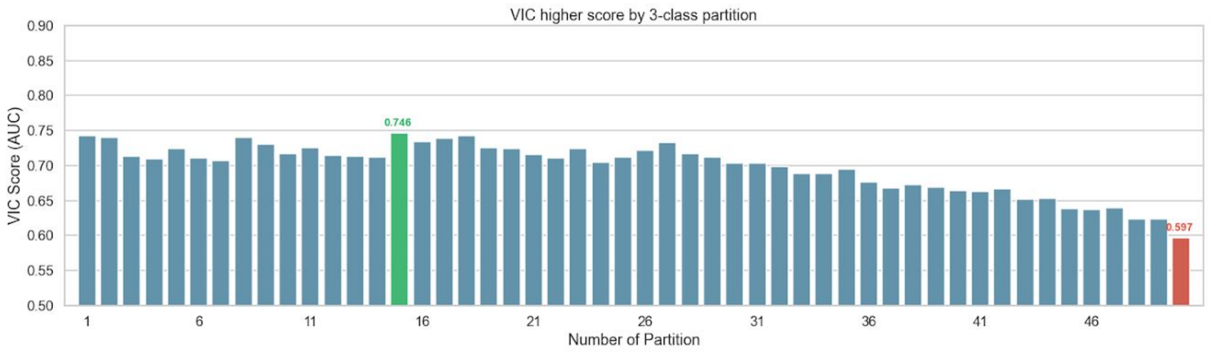

Figure 7. VIC higher score obtained for each partition.

Scores by algorithm

Figure 8 shows the AUC score obtained in each partition for each of the VIC classification algorithms. Furthermore, for each algorithm, it is shown which partition obtained the highest and the lowest AUC score. The algorithms, except for Decision Tree, followed the same trend as the overall VIC highest score graph, in which the best AUC score was found in the first partitions, while the worst was kept within the last ones. Moreover, it was found that Random Forest obtained the highest AUC score values for the best and worst partition in the overall VIC highest score graph. The worst performing classifier was Decision Tree, which obtained low AUC score values, compared to other classifiers.

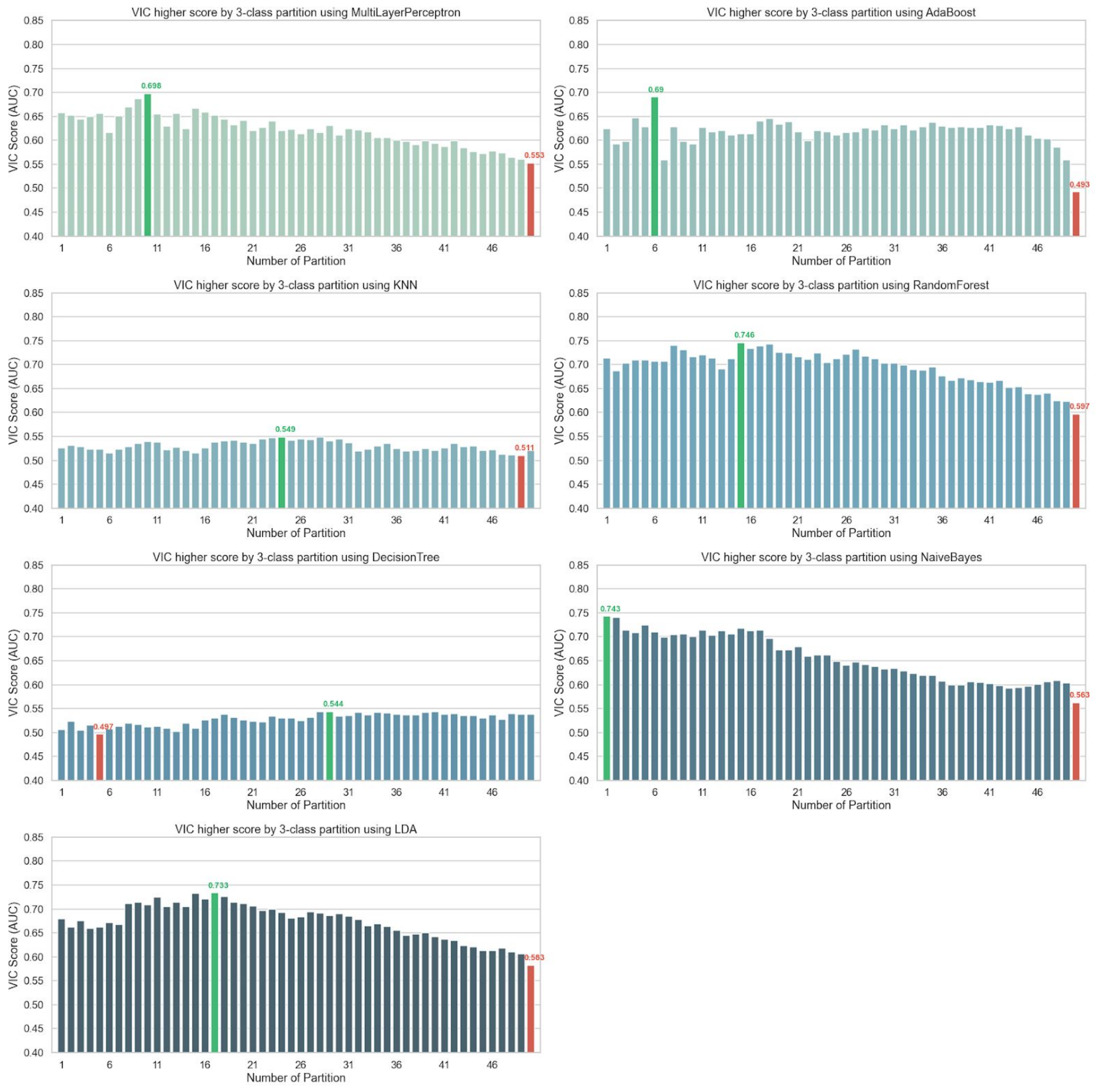

Figure 8. VIC higher score for each partition per classifier.

MVP

Figure 9 shows the number of times each algorithm obtained the highest AUC score in a particular partition. Random Forest was the algorithm that obtained the best AUC score in most partitions, 43 of the 50. It was followed far behind by Naive Bayes and LDA, with 5 and 2 partitions, respectively. On the other hand, Multilayer Perceptron, Adaboost, K-NN, and Decision Tree did not obtain the best AUC score in any partition. Like the two-clusters scenario, this performance suggests that these classification algorithms are not the most suitable for the dataset classification, or that chosen hyper-parameters are not optimal.

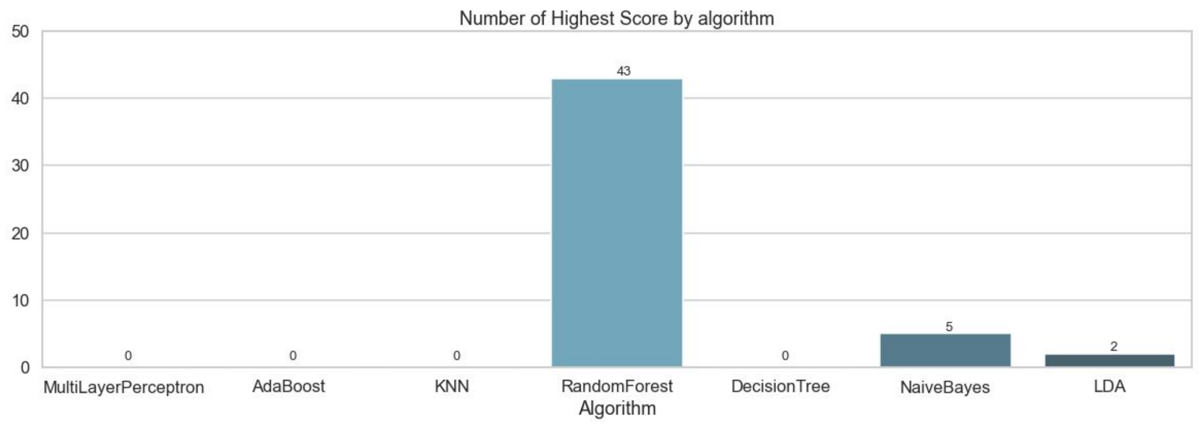

Figure 9. Number of VIC higher scores obtained per classifier.

Variance analysis by algorithm

Figure 10 shows a box plot that allowed us to analyze better the AUC score obtained on average by each algorithm. As expected, Random Forest obtained the highest average AUC score, being high above from the other algorithms. It was followed by LDA and Naive Bayes, who also obtained the best AUC score for some partitions.

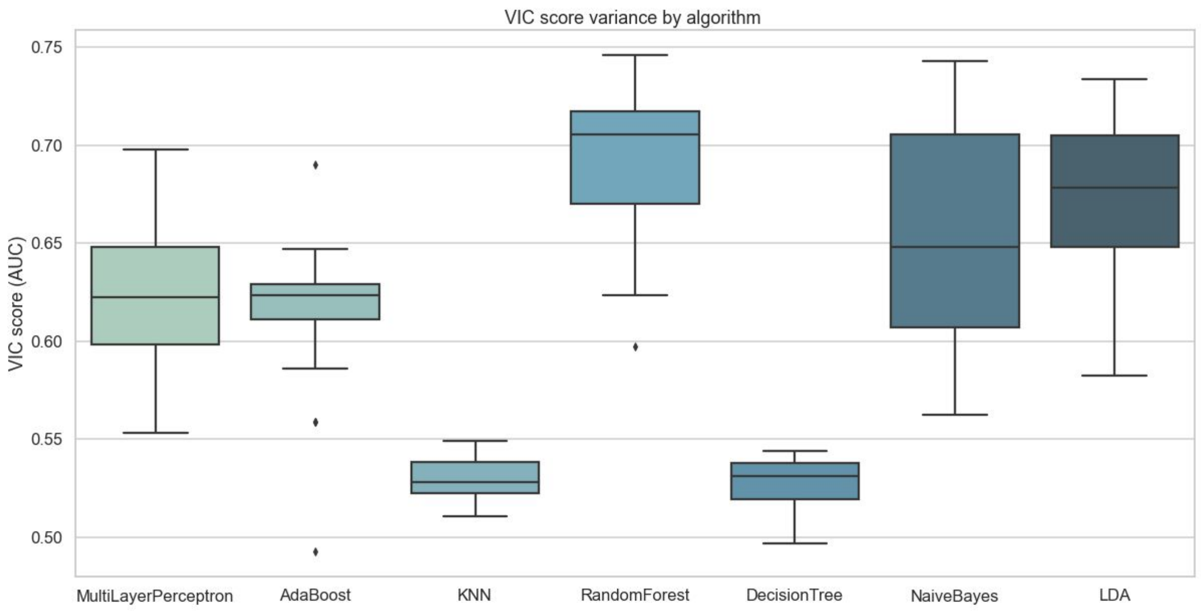

Figure 10. VIC score variance per classifier.

Conclusions

In this work, we evaluated the performance of the VIC Validity Index using a dataset with 50 two-cluster partitions and 50 three-cluster partitions. For the two-clusters partitions, it was found that the best partition (according to AUC) is the 50th with a score of 0.838 using the LDA algorithm. In the case of the three-clusters partitions, it was found that the best partition is the 15th with a score of 0.746 using the Random Forest algorithm. Another significant result is the opposite behavior observed in the Random Forest algorithm, which did not obtain the highest score in any partition for the two-clusters scenario, but completely dominated the three-clusters scenario obtaining the highest score in 95% of the partitions. A final finding is that K-NN and DT were consistently the worst algorithms in both scenarios. For future works, exploring other missing values techniques and the fine-tuning of each algorithm's hyper-parameters to improve the classification capacity are considered.

VIC Implementation in MATLAB

As part of this assignment, and further expanding this validity index's potential application into new platforms, a new implementation of the VIC algorithm was carried out in MATLAB. The project is public and freely accessible and can be found at https://github.com/jcarlosangelesc/VIC-ValidityIndex.

The current implementation allows the use of six different classifiers from which the user can freely pick. The available classifiers are:

1. Random Forest
2. K-Nearest Neighbors
3. Decision Trees
4. Linear Discriminant Analysis
5. Naive Bayes
6. Support Vector Machine

For instructions on how to add new classifiers and run the algorithm on a different dataset, please refer to the documentation available in the GitHub link provided above.

A notable advantage of this implementation is that it supports multithreading by default, so the computation time is significantly reduced.

References

[1] Rodríguez, J., Medina-Pérez, M. A., Gutierrez-Rodríguez, A. E., Monroy, R., & Terashima-Marín, H. (2018). Cluster validation using an ensemble of supervised classifiers. Knowledge-Based Systems, 145, 134-144.

[2] VIC. https://sites.google.com/site/miguelmedinaperez/software/vic. Retrieved September 4, 2020.