Practical Application 1
Machine Learning Assignment

David Cabornero Pascual
david.cabornero@alumnos.upm.es

June 18, 2022

1 Introduction

In this practice, a Dataset of the Sloan Digital Sky Survey will be used to predict which type of heavenly body are we watching. In fact, there are three types of bodies that we will predict: stars, galaxies, and quasars.

The main objective of this practice is to implement 5 algorithms (Neural Network, k-Nearest Neighbors, Rule Induction, Support Vector Machines and Classification Trees) and analyze them in 4 cases:

- With all variables.
- With a univariate filter feature subset selection.
- With a multivariate filter feature subset selection.
- With a multivariate wrapper feature subset selection.

After that, we have to analyze results like accuracy or execution time and extract some conclusions about them. Additionally, we also have to see the behavior of the algorithms when it's possible (for example, we can analyze the tree created by the Classification Tree to see its behavior).

2 Problem Description

The Dataset [2] trained in this Assignment is from the Sloan Digital Sky Survey, and the mean goal is to predict with some features whether the telescope is seeing a Star, Galaxy or Quasar. For this purpose, we have 10.000 instances with 18 parameters. Let's proceed to make a brief description of them:

- *objid*, *ra* and *dec* specifies the object seen and its position in the sky.
- *u*, *g*, *i*, *r* and *z* corresponds with the Thuan-Gunn magnitude system. They represent the behavior of the 5 telescope bands.
- *run*, *rerun*, *camcol* and *field* represents where is the heavenly body in the photo.

- *specobjid* is an object identifier.
- *redshift* is related to the wavelength of the radiation.
- *plate* is the ID of the plate used in the telescope.
- *MJD* is the date when the photo was taken.
- *fiberID* indicates the ID of the fiber in the spectrograph.

3 Methodology

In contrast with other kind of Machine Learning assignments, now we are interested in analyzing certain prediction models and how feature selection affects them. For this reason, an exhaustive Grid Search is not going to be carried out, we are only interested in doing *honest* models. Besides, we can only use the train set in the *Feature Filter* and the *Wrapper*. That's why the Dataset has been split into a permanent train set and a permanent test set that has been used for all the experiments. In fact, the test set is a 10% of the Dataset, in other words 1.000 samples.

On the other hand, a good preprocessing is necessary, despite the fact that the dataset is very clean. In the first place, some nominal parameters had been read as numerical parameters. We are talking about:

- *objid*: ID of each sample
- *fiberid*: ID of the optical fiber
- *specobjid*: ID of the object identified
- *plate*: ID of the plate used in the telescope
- *run*: ID of each scan

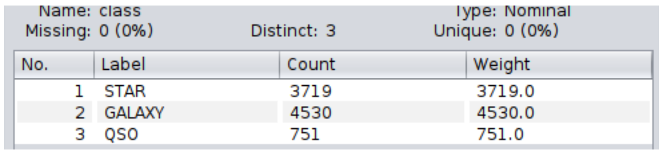
Moreover, we have to remove some of these variables even before applying any filter. *objid* doesn't give any information, is only the ID of each sample. Respect to *fiberid*, *specobjid* and *plate*, there are more than 1.000 values for each feature, so we must remove them too to avoid overfitting (specially in the Neural Network). Consequently, our unique nominal variable will be *run*, whose 23 unique values are admissible.

Regarding normalization, there are some variables with values much bigger than others (for example, *mjd* has mean 52.900 and *camcol* only has values between 1 and 6). With that values, maybe some algorithms don't converge (i.e. Neural Networks) and others can show poorer results (i.e. KNN). For this reason, all numerical values have been normalized to guarantee a correct optimization.

For all the assignment tasks Weka has been used, except for normalization of the train and test set. For that task, Python with Pandas and NumPy has been used.

We have to fill in a table with the prediction algorithms and distinct feature subset selection. The accuracy is the best metric to show the performance, but the three classes are unbalanced:

For this reason, the confusion matrix will be used too, specially when accuracy shows the best results. Another advantage about feature selection is the execution time reduction.



Name: class | Type: Nominal
Missing: 0 (0%) | Distinct: 3 | Unique: 0 (0%)

No.	Label	Count	Weight
1	STAR	3719	3719.0
2	GALAXY	4530	4530.0
3	QSO	751	751.0

Figure 1: Number of instances of each class

Because of that, a table with execution times (the sum of train and test) is given with the another two.

Now, the algorithms chosen and their parameters will be shown. All of them have been chosen after seeing that they are *honest*, i.e. they show relatively good results.

3.1 Prediction algorithms

Firstly, we have the classic **kNN** algorithm. The decision was taking account of the nearest 20 neighbors, and Euclidean distance was used. Weighting or rejection weren't used.

Secondly, we have the **Rule Induction** algorithms. In this case, RIPPER algorithm has been chosen. This algorithm hasn't got almost any parameters to modify, so in this case we have an easy configuration.

With **Support Vector Machines**, a non-lineal one has been used. In fact, it has a polynomial kernel with exponent 2.

Now we have **Neural Networks**, In fact the Multilayer Perceptron. In this case, the neural network has a hidden layer with 32 neurons, as many problems of prediction can be solved with only one hidden layers. Nominal values have been transformed into binary values with the method *nominalToBinary*, similar to *OneHotEncoder* in *scikit-learn*. The transfer function is the sigmoid one and the rest of parameters are set by default.

In the last place, we have **Classification Trees**, In fact the C4.5 algorithm. This algorithm has not many parameters to choose, it's an ID3 algorithm that can handle with missing and continuous values and uses *gain ratio* to split [1].

3.2 Feature subset selection

For univariate filters, the **gain ratio** method has been used. The threshold has been set in 0.1, i.e., if an attribute has a ratio of less than 0.1 it will be discarded. For the multivariate filter, **CFS** algorithm has been used in this case. Lastly, the **Wrapper** method provided by *Weka* has been used. The metric provided to evaluate is accuracy for discrete attributes and RMSE (Root Mean Square Error) for continuous attributes.

4 Results

For a start, we can see the results of each feature subset selection in Table 1, while tables about accuracy and time are shown in Table 2 and Table 3.

	Univar.	Multivar.	kNN	RIPPER	SVM	NN	C4.5
ra					x		
dec					x		
u			x		x	x	x
g			x		x	x	x
r	x				x	x	x
i	x				x		x
z	x				x		x
run					x		
rerun							
camcol					x		
redshift	x	x	x	x	x	x	x
mjd	x	x			x		
field							x

Table 1: Attributes chosen with univariate filter, multivariate filter and with each wrapper filter.

	kNN	RIPPER	SVM	NN	C4.5
All variables	0.85	0.988	0.944	0.964	0.979
Univariate	0.946	0.985	0.868	0.975	0.98
Multivariate	0.973	0.985	0.813	0.978	0.98
Wrapper	0.985	0.988	0.944	0.979	0.985

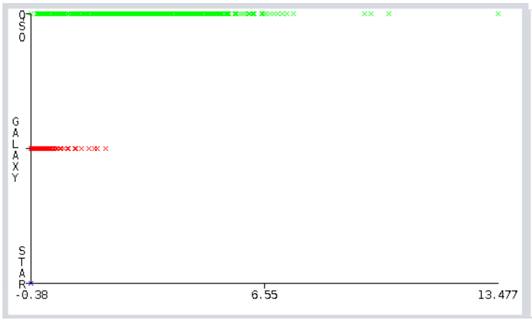
Table 2: Table with accuracies.

	kNN	RIPPER	SVM	NN	C4.5
All variables	0.83	0.57	64.15	81.98	0.28
Univariate	0.37	0.31	21.56	21.69	0.05
Multivariate	0.44	0.17	25.55	15.58	0.02
Wrapper	0.78	0.87	56.18	18.97	0.75

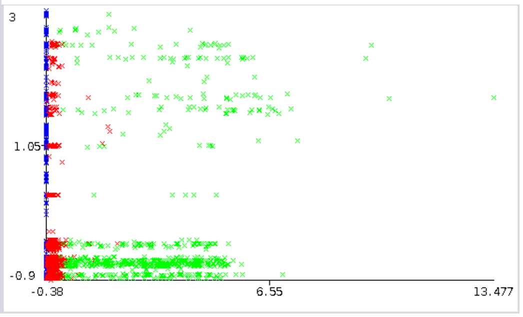
Table 3: Table with execution times (in seconds).

Regarding the visualization of algorithms, an attempt to show how it works has been carried out. Starting with **k-Nearest Neighbors**, it's impossible to understand the distance with a graphical model with more than 3 dimensions. For this reason, we can only understand how *KNN* works with the *CFS* algorithm and the *wrapper* with Figure 2.

The **RIPPER** algorithm is very understandable. We have a sequence of decision rules,



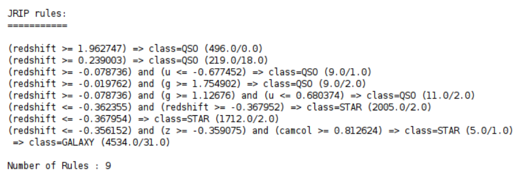
(a) *redshift* in X, *class* in Y



(b) *redshift* in X, *mjd* in Y

Figure 2: Visualization of classes. In red, *Galaxy*; in dark blue, *Star*; in green, *Quasar*.

so we can see directly how the algorithm is working inside. We can see the rules in each case in Figure 3.

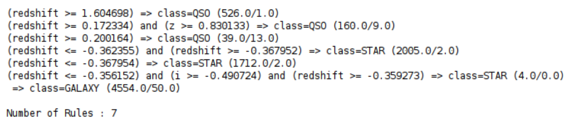


```
JRIP rules:
===========

(redshift >= 1.962747) => class=QSO (496.0/0.0)
(redshift >= 0.239003) => class=QSO (219.0/18.0)
(redshift >= -0.078736) and (u <= -0.677452) => class=QSO (9.0/1.0)
(redshift >= -0.019762) and (g >= 1.754902) => class=QSO (9.0/2.0)
(redshift >= -0.078736) and (g >= 1.12676) and (u <= 0.680374) => class=QSO (11.0/2.0)
(redshift <= -0.362355) and (redshift >= -0.367952) => class=STAR (2005.0/2.0)
(redshift <= -0.367954) => class=STAR (1712.0/2.0)
(redshift <= -0.356152) and (z >= -0.359075) and (camcol >= 0.812624) => class=STAR (5.0/1.0)
 => class=GALAXY (4534.0/31.0)

Number of Rules : 9
```

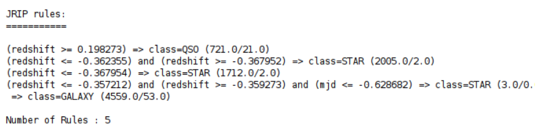
(a) All variables.



```
(redshift >= 1.604698) => class=QSO (526.0/1.0)
(redshift >= 0.172334) and (z >= 0.830133) => class=QSO (160.0/9.0)
(redshift >= 0.200164) => class=QSO (39.0/13.0)
(redshift <= -0.362355) and (redshift >= -0.367952) => class=STAR (2005.0/2.0)
(redshift <= -0.367954) => class=STAR (1712.0/2.0)
(redshift <= -0.356152) and (i >= -0.490724) and (redshift >= -0.359273) => class=STAR (4.0/0.0)
 => class=GALAXY (4554.0/50.0)

Number of Rules : 7
```

(b) Univariate Filter

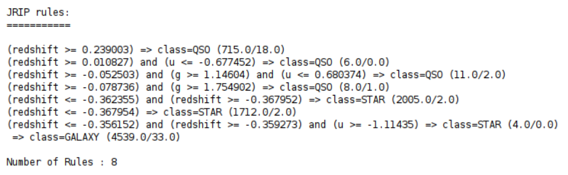


```
JRIP rules:
===========

(redshift >= 0.198273) => class=QSO (721.0/21.0)
(redshift <= -0.362355) and (redshift >= -0.367952) => class=STAR (2005.0/2.0)
(redshift <= -0.367954) => class=STAR (1712.0/2.0)
(redshift <= -0.357212) and (redshift >= -0.359273) and (mjd <= -0.628682) => class=STAR (3.0/0.
 => class=GALAXY (4559.0/53.0)

Number of Rules : 5
```

(c) Multivariate filter



```
JRIP rules:
===========

(redshift >= 0.239003) => class=QSO (715.0/18.0)
(redshift >= 0.010827) and (u <= -0.677452) => class=QSO (6.0/0.0)
(redshift >= -0.052503) and (g >= 1.14604) and (u <= 0.680374) => class=QSO (11.0/2.0)
(redshift >= -0.078736) and (g >= 1.754902) => class=QSO (8.0/1.0)
(redshift <= -0.362355) and (redshift >= -0.367952) => class=STAR (2005.0/2.0)
(redshift <= -0.367954) => class=STAR (1712.0/2.0)
(redshift <= -0.356152) and (redshift >= -0.359273) and (u >= -1.11435) => class=STAR (4.0/0.0)
 => class=GALAXY (4539.0/33.0)

Number of Rules : 8
```

(d) Multivariate wrapper

Figure 3: Rules induced from RIPPER algorithm.

In contrast, **SVM** algorithm is too opaque to understand how it works inside. In all the cases we have thousands of support vectors, so we can't understand at all what is happening there.

In the same way, the problem is repeated in **Neural Networks**. There are hundreds of links between neurons, each one with a weight. We can imagine the quantity of links seeing the first Neural Network, with all the features remaining in Figure 4.

In the last place, we have the Classification Tree **C4.5**. This algorithm generates a tree of decisions that ends with a certain classification, so we can see easily how it classifies the samples. Let's see the four trees in Figure 5.

To conclude this section, the confusion matrices of the best four classifiers will be shown.

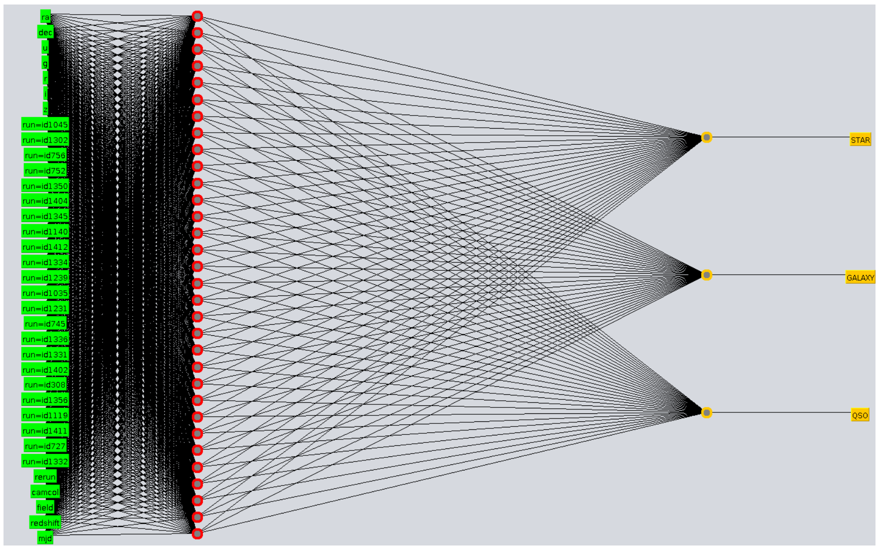


Figure 4: Graphical representation of the Neural Network with all features.

This information is to know more about which classes are being bad predicted, since they are unbalanced. This can be shown in Figure 6.

5 Discussion

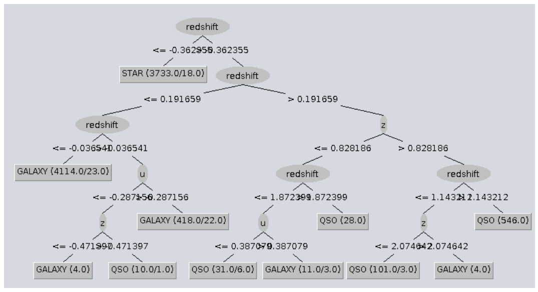
In this case this Dataset has very good results with a vast majority of predictors, exceeding the 95% in many cases. Just because the same train and test set has been used in all cases, small improvements in the tables are significant.

Let's start analyzing the features chosen with each selector. Regarding Table 1, we can infer two immediate conclusions: the relevance of the parameter *redshift* and the irrelevance of the parameter *rerun*. In the first case, we can see easily what's happening: one of the majoritarian classes is easy to predict having only this attribute, and the another two can be predictable with a little error. In the second case, this variable has only one value: 301. It shows how the image has been processed, but it's useless in this dataset.

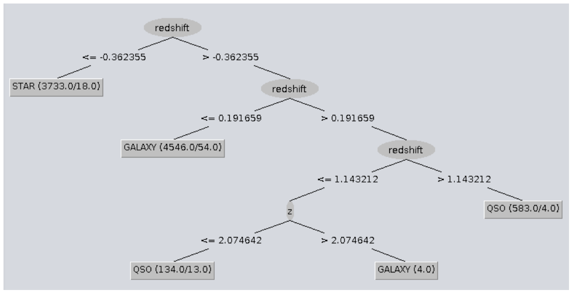
Now, we are going to analyze the predictive algorithms individually. Starting with **kNN**, there are too dimensions in the Dataset to see how is the space and why kNN is working. Nonetheless, we can see the best improvement of the table when we reduce the number of attributes. Clearly, unnecessary features are noise in this case.

In the beginning of the section, we have seen why using only *redshift* works nicely (*Wrapper* case), but we can analyze also the *multivariate filter* case, with *redshift* and *mjd*. We can see in Figure 2 that a method like kNN works really well in this plane: the points of each class are condensed in one certain place, generally.

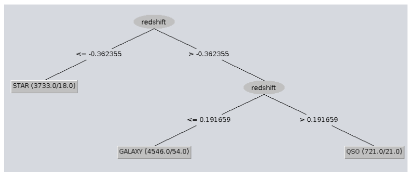
Secondly, let's analyze the **RIPPER** algorithm. This algorithm is much more transparent: we can see which are the rules that decided which class is chosen. Surprisingly, in



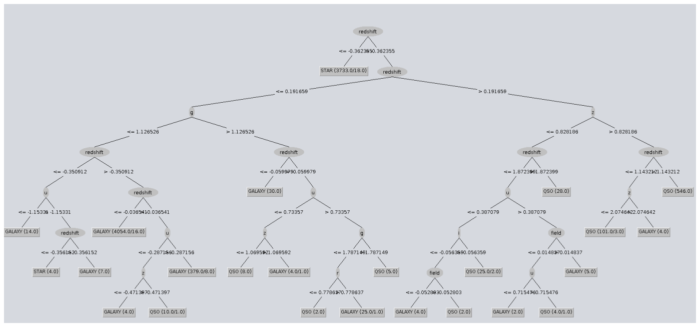
(a) All variables.



(b) Univariate Filter



(c) Multivariate filter



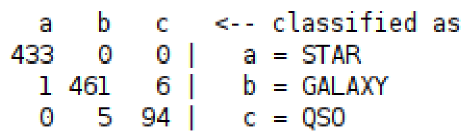
(d) Multivariate wrapper

Figure 5: Decision Trees created with C4.5 algorithm.

the worst case the algorithm needs only 9 rules to infer a method with more than 98% of accuracy. This algorithm doesn't improve more with the feature filters, so it's not sensitive to redundancy or irrelevance of features. The most curious case is the Multivariant filter, when with only 5 rules and 2 attributes (one of them only used in one rule) we have almost the best accuracy with one of the most little times, letting us see that we have a very simple problem with a lot of noise.

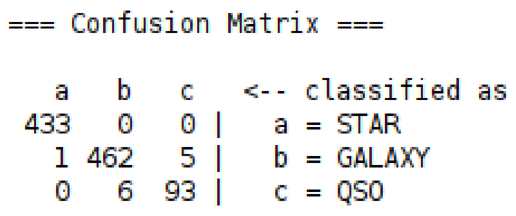
Now we have the **Multilayer Perceptron**. It's the less transparent method, since we have hundreds of links between neurons that can't be analyzed individually. Although, we can infer that we are getting very nice results thanks to a lot of time. While kNN, RIPPER and C4.5 don't need more than 1 second to train and test, a Neural Network needs 15 seconds in the best case (with two features), and 82 seconds in the worst (with all features). It's possible that 32 neurons in the hidden layer is too much considering that in the *Multivariate Filter* case we are working with only two numerical parameters, so probably we are experimenting overfitting and an increase of time because of that. With this neural network, we can conclude that a more exhaustive analysis should be carried out if we want to get the optimal solution with an MLP.

In the case of **SVM**, it's again too opaque with thousands of support vectors. This is the only case which the accuracy decreases significantly with the *Univariate and Multivariate Filters*, but it recovers when we use the *Wrapper*. Concretely, the *Wrapper* doesn't do



```
  a   b   c   <-- classified as
433   0   0 |   a = STAR
  1 461   6 |   b = GALAXY
  0   5  94 |   c = QSO
```

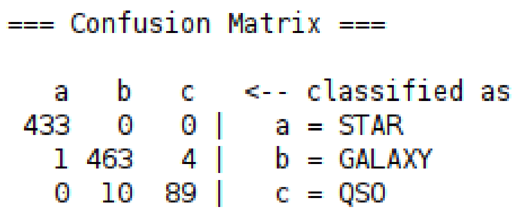
(a) RIPPER, all variables



```
=== Confusion Matrix ===

  a   b   c   <-- classified as
433   0   0 |   a = STAR
  1 462   5 |   b = GALAXY
  0   6  93 |   c = QSO
```

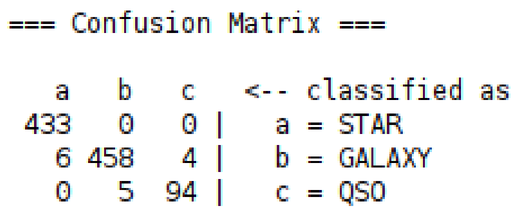
(b) RIPPER, Wrapper



```
=== Confusion Matrix ===

  a   b   c   <-- classified as
433   0   0 |   a = STAR
  1 463   4 |   b = GALAXY
  0  10  89 |   c = QSO
```

(c) kNN, Wrapper



```
=== Confusion Matrix ===

  a   b   c   <-- classified as
433   0   0 |   a = STAR
  6 458   4 |   b = GALAXY
  0   5  94 |   c = QSO
```

(d) C4.5, Wrapper

Figure 6: Confusion matrices of the best classifiers.

almost nothing, since it only removes two parameters. We can conclude in this case that SVM needs most of the attributes in order to avoid overfitting, probably. Again, this classifier takes more or less one minute on doing a training properly, so it's clear that the polynomial kernel with exponent 2 was too complex for this problem.

Finally, the algorithm **C4.5** is one of the most transparent. It develops a tree with a small number of branches, which has more or less a 98% of accuracy. Maybe the most interesting case is the *Wrapper*, when we obtain the most complex tree, even more than the first one with all variables. We can deduce that this algorithm is sensitive to the attribute reduction in some cases, being able to do more accurate trees. Although, there is another curious case with the *Multivariate Filter* case. The algorithm is able to get a 98% of accuracy only using two rules based on *redshift*. With that, we can see again how easily we can solve this problem with a very good accuracy.

Finally, we are going to analyze the confusion matrices of the best classifiers exposed in Figure 6. In them, we can see that all the Stars are predicted properly, and the main problem is between Quasars and Galaxies, that are confused in a balanced way. An exception can be *kNN*, where most Quasars are classified as Galaxies (and not vice versa). Although, we can conclude that the minority of the Quasar class is not a remarkable problem.

6 Conclusion

In this assignment we have worked with a *Dataset* with 18 features and 10.000 samples that tries to predict a class between three: *Galaxy*, *Quasar* and *Star*. Besides the removal of some nominal variables which has too many values, only a few variables are enough to predict correctly a great part of the *Dataset*. Because of that, the filters and the wrapper becomes very illustrative and useful.

In fact, with only *redshift* and *mjd* are enough to obtain results above 97%. Because of that, simpler and more transparent classifiers has been the winners: *RIPPER*, *kNN*, and *C4.5*. In contrast, those classifiers that usually can solve huge and difficult problems like SVM and Multilayer Perceptron has worse results, probably because of overfitting.

Regarding the classifiers deeply, kNN is more sensitive to unnecessary variables, showing the biggest improvement when they were removed. However, SVM needed that worthless variables, probably to avoid the overfitting caused by a too complex kernel. This predictor could be better with a simpler kernel, just as the Neural Network, but adjusting the parameters of classifiers to get the best accuracy was not a part of this assignment. Nonetheless, it's an interesting part, and it remains as future work.

In conclusion, we have a clear *dataset* which can be predicted properly with only two variables, and works really well with the simplest classifiers. The *wrapper* and the multivariate filter applied to these predictors shows the best accuracy, reaching a 98%. Sometimes, a huge Neural Network or a very sophisticated SVM are not a good solution in Machine Learning.

References

[1] C. Bielza Lozoya and P. Larrañaga Mújica. *Data-driven computational neuroscience : machine learning and statistical models.* Cambridge University Press, Cambridge, 2020.

[2] L. Grosser. Sloan digital sky survey, 03 2018.

[3] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, San Francisco, 2nd edition, 2005.