Binary Classification via Support Vector Machine: QSO vs STAR

Luca Sannino

Università degli Studi di Napoli FEDERICO II

- Summary:

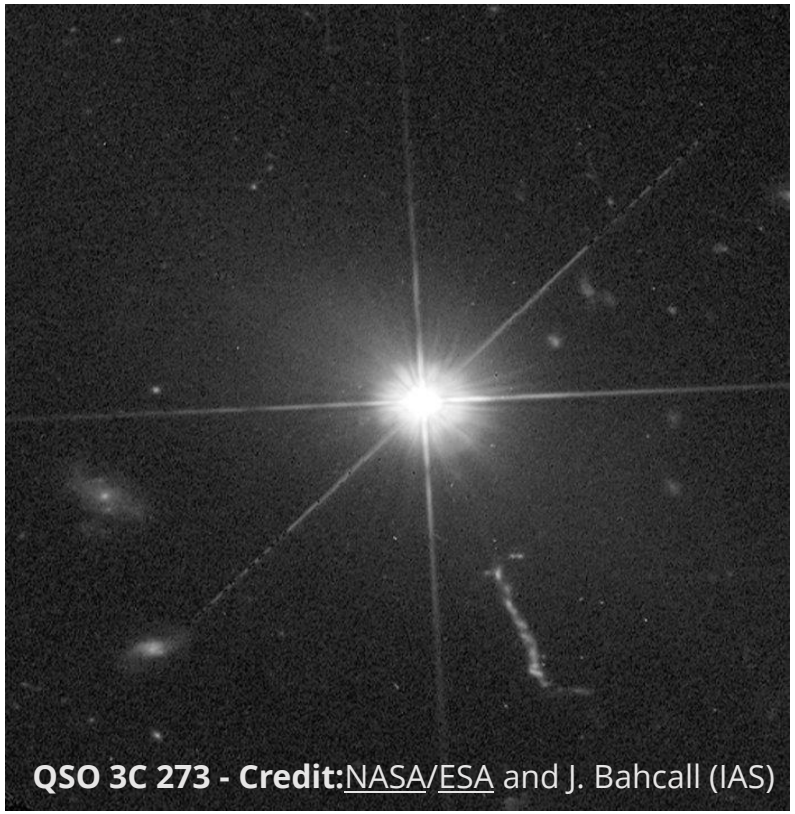
- Theoretical descriptions:
  - Phisical differences between qso and stars.
  - Support Vector Machine.

- Data Selection and pre-processing:
  - SLOAN and SQLs comands.
  - Data skimming through FLAGS.
  - EDA.

- Support Vectore Machine:
  - Model definition.
  - Training and test.

- Performance and Conclusion:
  - Confusion Matrix.
  - Usefull Statistical Parameter.

Theoretical descriptions : QSO vs STAR



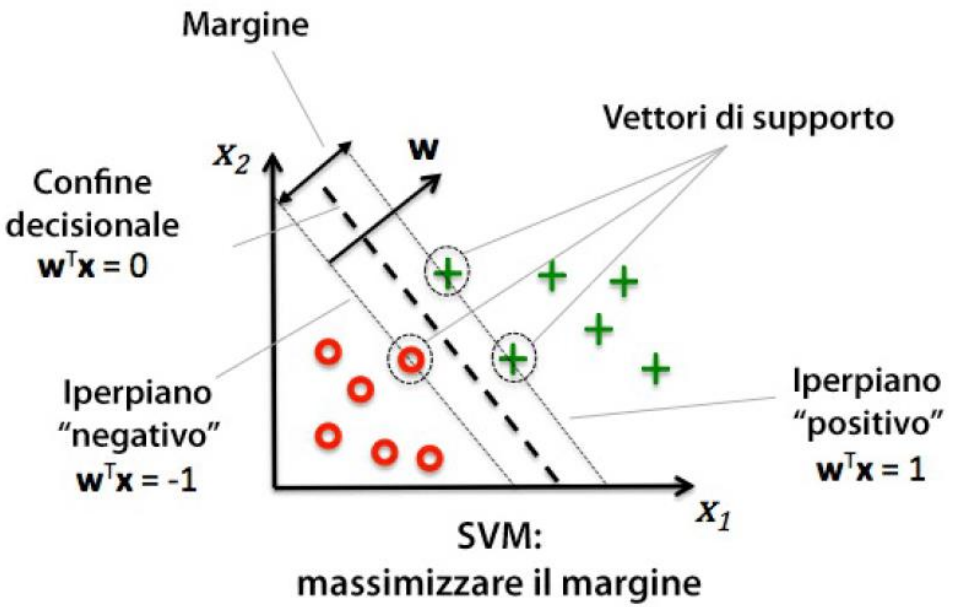
QSO 3C 273 - Credit:NASA/ESA and J. Bahcall (IAS)

WHITE DWARF STAR STEIN 2051 B **- Credit:**NASA, ESA, and K. Sahu (STScI)

- ★☆☆ Strong resemblance in astronomical images, but really different object.
- ★★☆ QSOs are very luminuos AGN powered by a supermassive BH.
- ★★★ QSO's luminosity, mass and dimension are much larger than stellar's one.

Theoretical descriptions : Support Vector Machine

- Data are linearly separable → two parallel hyperplanes detecting the margin
- Hyperplane in the halfway between positive and negative hyperplanes is the maximum-margin
- Support Vectors → data used that defined the margin.
- The SVM aim is maximize the margin.
- A SVM assigns sample to category based on which side of Hyperplane data falls.



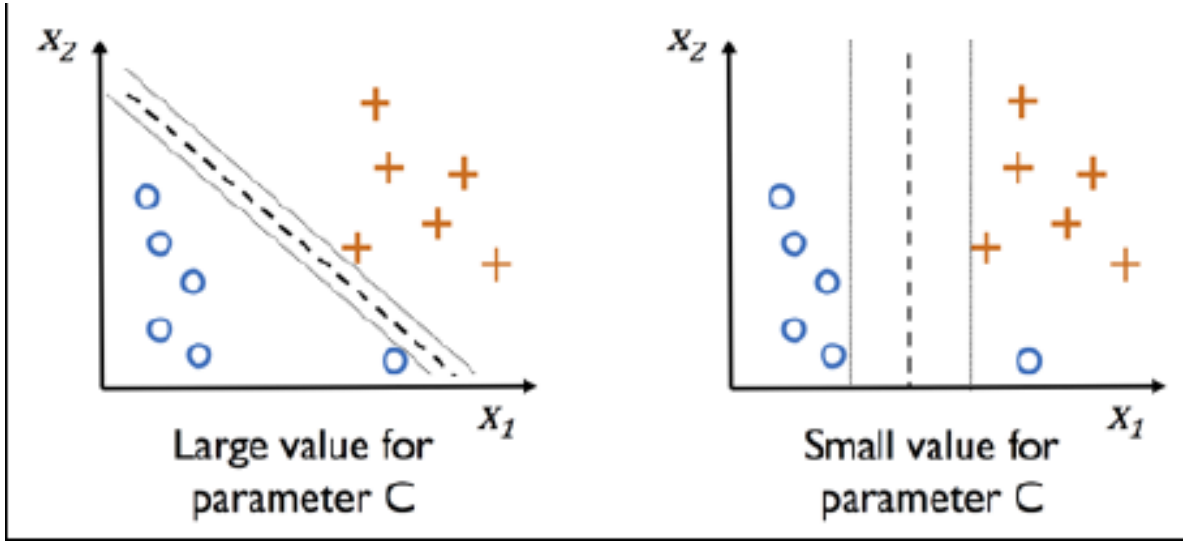
Geometrically the two hyperplanes are defined by: $\mathbf{w}^T\mathbf{x} - b = 1$ $\mathbf{w}^T\mathbf{x} - b = -1$ → The distance is: $\frac{2}{\|\mathbf{w}\|}$

This is the value to optimize to find the maximum margin, there is tight constraints

Theoretical descriptions : Support Vector Machine

However is computationally easier minimize the modified equation in which ξ is called slack variable and are the distances of support vectors from the neg. And pos. hyperplane

$$\frac{1}{2}||w||^2 + c\sum_{i=1}^{m}\left(\xi_1 + \xi_i^*\right)$$



The c variable control penality for missclassification

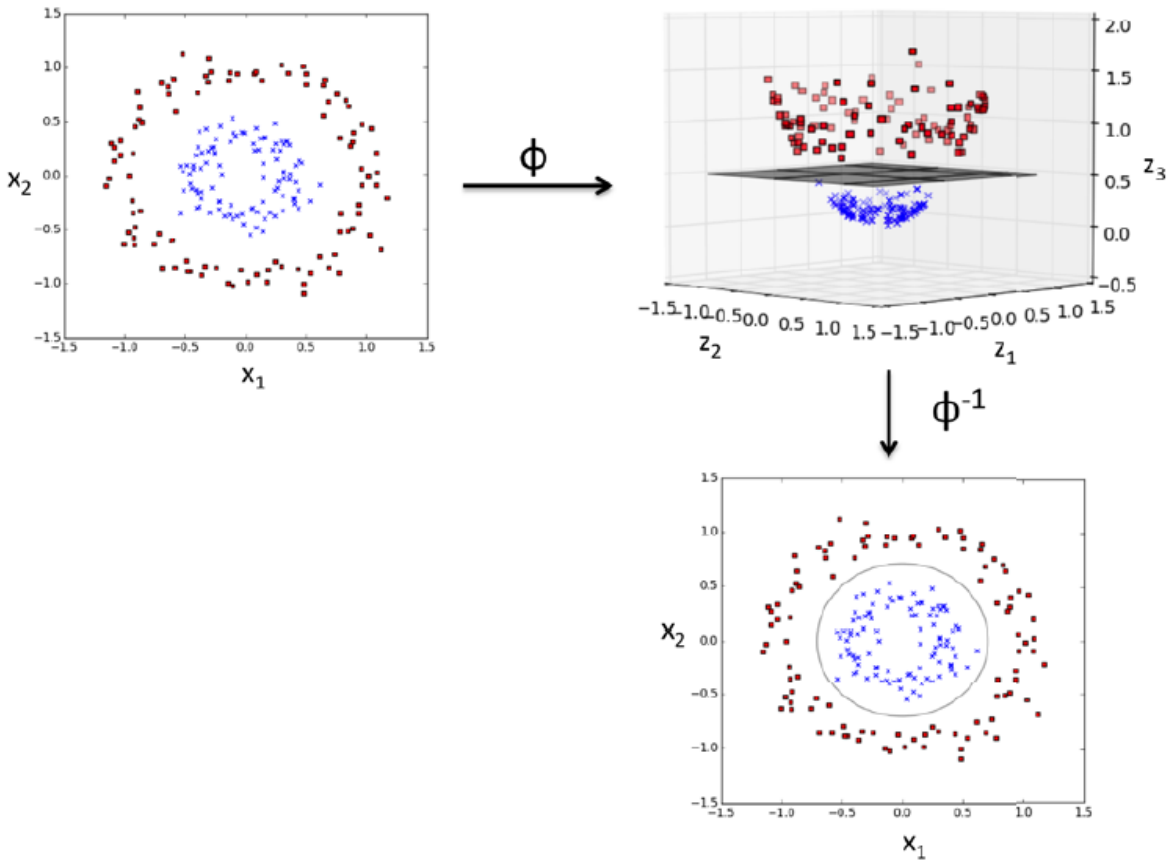
Control c variable → tune Bias/Variance trade-off.

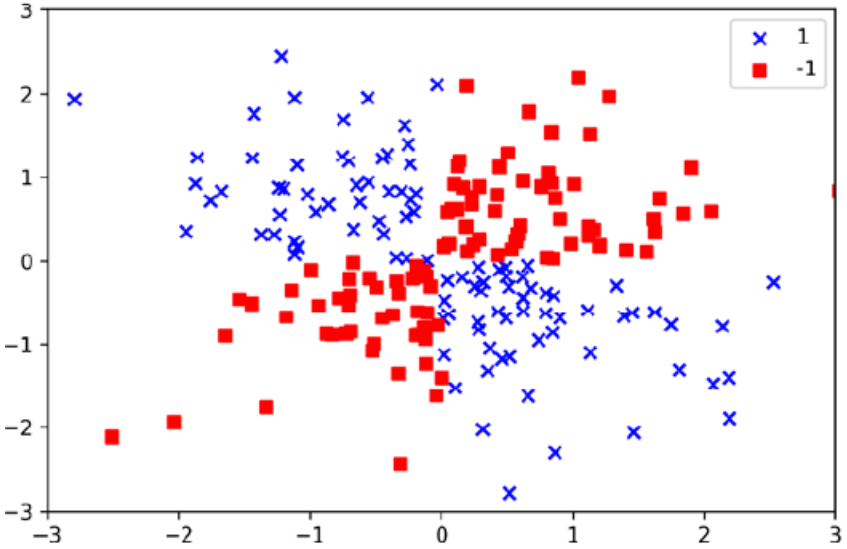
Decreasing c → Increasing Bias → Lowers Variance.

Theoretical descriptions : Support Vector Machine

Data not linearly seprable in a space → the original space is mapped in a higher dimension space (Boser et. Al 1992) to make separation easier.

So we using the Kernel Method.





The Kernel Method create a nonlinear combinations of the features to project data in higher dimension through a mapping function ϕ where it's becomes linearly separable.

- Polinomial Kernel.
- Radial Basis Function (or Gaussian kernel).

Theoretical descriptions : Support Vector Machine

Kernel Methods could bring to high computational cost if it weren't thanks to Kernel Trick.

Kernel Trick allows to operate in the original space without computing the transofrmation

Here's an example

$$\mathbf{x} = (x_1, x_2, x_3)^T$$
$$\mathbf{y} = (y_1, y_2, y_3)^T$$

We want to map x and y in a 9-dimensional space

$$\phi(\mathbf{x}) = (x_1^2, x_1x_2, x_1x_3, x_2x_1, x_2^2, x_2x_3, x_3x_1, x_3x_2, x_3^2)^T$$
$$\phi(\mathbf{y}) = (y_1^2, y_1y_2, y_1y_3, y_2y_1, y_2^2, y_2y_3, y_3y_1, y_3y_2, y_3^2)^T$$

That is a dot product with a high computational complexity

$$\phi(\mathbf{x})^T\phi(\mathbf{y}) = \sum_{i,j=1}^{3} x_i x_j y_i y_j$$

With the kernel function we need to compute the dot product between x transpost and y.
We have a lower level of computational complexity.

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= (\mathbf{x}^T\mathbf{y})^2 \\ &= (x_1y_1 + x_2y_2 + x_3y_3)^2 \\ &= \sum_{i,j=1}^{3} x_i x_j y_i y_j \end{aligned}$$

Theoretical descriptions : Support Vector Machine

❖ *PROS of SVM*

- The solution will be a global minimum.
- Usable for linearly and non linearly separable data.
- Highly costumizable thanks the possibility to choice among different kernels

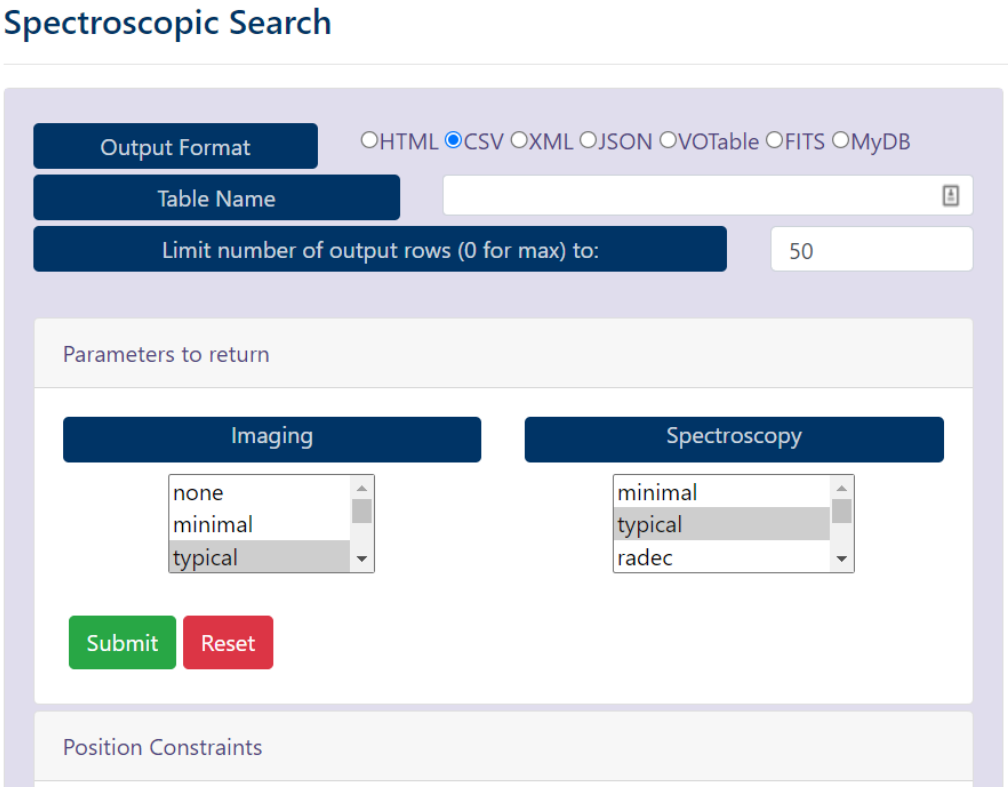
❖ *Cons of SVM*

- Can’t handle text structures
- Can be highly computationally speaking, expensive for a large set of data.
- Choice of Kernel is one of the strongest feature but also the greatest limitation.
- The model tries to maximaze separation between classes, too much data overlap could make it difficult to find a good solution

Data Selection and pre-processing : SLOAN and SQLs command

First step of the work was to obtain SLOAN's data.

Spectroscopic Search is the easiest way to obtain data but has a low capacity of costumization and control over the data.



https://skyserver.sdss.org/dr18/SearchTools/SQS

Through SQL search we can explicty ask from which tables we wants data, which category and that we don't want any duplicates (mode = 1)

```
SELECT TOP 10000
s.plate,s.mjd,s.fiberid,s.z,s.zErr,s.zWarning,s.class,p.run,p.rerun,p.camCol,p.field,p.obj,cast(str(p.ra,13,8) as float) as ra,
cast(str(p.[dec],13,8) as float) as dec,p.r,p.g,p.i,p.u,p.z,p.flags_r
FROM ..SpecObj as s
JOIN ..PhotoObj AS p ON s.bestObjID = p.objID
WHERE  (s.class = 'QSO') AND ( p.type = 3 OR p.type = 6) AND mode = 1
```

Data Selection and pre-processing : Skimming Data Through flags

Obtained the csv files, and created the dataframes, i had to select only qso and star with a clean photometry thanks the SLOAN guidlines.

	plate	mjd	fiberid	z	zErr	zWarning	class	run	rerun	camCol	field	obj	ra	dec	r	g	i	u	z1	flags_r
0	269	51910	319	0.353129	0.000054	0	QSO	756	301	3	232	39	150.565980	-0.182256	18.79082	18.99710	18.79543	19.25479	18.17451	4503599895838736
1	6714	56447	754	2.473536	0.000272	0	QSO	1345	301	1	533	232	232.418487	54.057916	18.97910	19.11830	18.88298	19.74728	18.62487	4503874773745680
2	6715	56449	352	0.725679	0.000187	0	QSO	1345	301	1	552	148	235.882307	52.226394	19.85699	19.91018	19.79552	20.27558	19.58408	4503874773712896
3	8276	57067	700	1.104225	0.000445	0	QSO	1350	301	6	89	548	116.899522	43.922496	20.48068	20.84040	20.43517	20.91058	20.30026	4503874773712896
4	8276	57067	698	1.394426	0.000749	0	QSO	1350	301	6	90	439	116.935499	44.052300	20.97504	21.22012	20.83448	21.44579	20.92607	4503874773712896
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	8061	58253	474	1.024839	0.000404	0	QSO	4629	301	2	258	316	246.041306	53.244278	20.40130	20.64433	20.46348	20.63321	20.22085	4503874773712896
9996	1808	54176	468	1.748756	0.000367	0	QSO	3841	301	4	299	242	209.870322	6.973265	18.98812	19.09812	18.75868	19.23244	18.67760	4503599895838720
9997	5487	55982	198	3.151740	0.000322	0	QSO	3836	301	2	526	520	228.359173	8.575237	20.29221	20.65932	20.18946	25.31788	20.06859	4503599895838736
9998	3367	54998	616	2.765463	0.001890	0	QSO	3699	301	6	68	48	195.356975	47.991871	19.28098	19.51024	19.20381	20.30505	19.16380	4503599895838720

So i focused my attention on the digits of flags_r and analyzed it through a python code:

```
df_quasar['flags_r'] = df_quasar['flags_r'].astype('int64')
df_clean_quasar = df_quasar.copy()
df_clean_quasar = df_clean_quasar[(df_clean_quasar['flags_r'] & 0x20 == 0) &
                                  (df_clean_quasar['flags_r'] & 0x80000 == 0) &
                                  (df_clean_quasar['flags_r'] & 0x400000000000 == 0) &
                                  (df_clean_quasar['flags_r'] & 0x800000000000 == 0) &
                                  (df_clean_quasar['flags_r'] & 0x10000000000 == 0) &
                                  ((df_clean_quasar['flags_r'] & 0x100000000000 == 0) | (df_clean_quasar['flags_r'] & 0x1000 == 0)) &
                                  (df_clean_quasar['flags_r'] & 0x10000000 != 0) &
                                  (df_clean_quasar['flags_r'] & 0x40000 == 0) &
                                  (df_clean_quasar['flags_r'] & 0x80 == 0) &
                                  (df_clean_quasar['flags_r'] & 0x4 == 0)]
df_clean_quasar
```

Data Selection and pre-processing : EDA

```
2      False  False  False    False  False  False      False  False  False
3      False  False  False    False  False  False      False  False  False
4      False  False  False    False  False  False      False  False  False
...      ...    ...    ...      ...    ...    ...        ...    ...    ...
18931  False  False  False    False  False  False      False  False  False
18932  False  False  False    False  False  False      False  False  False
18933  False  False  False    False  False  False      False  False  False
18934  False  False  False    False  False  False      False  False  False
18935  False  False  False    False  False  False      False  False  False

       rerun  ...  field    obj     ra    dec      r      g      i      u  \
0      False  ...  False  False  False  False  False  False  False  False
1      False  ...  False  False  False  False  False  False  False  False
2      False  ...  False  False  False  False  False  False  False  False
3      False  ...  False  False  False  False  False  False  False  False
4      False  ...  False  False  False  False  False  False  False  False
...      ...  ...    ...    ...    ...    ...    ...    ...    ...    ...
18931  False  ...  False  False  False  False  False  False  False  False
18932  False  ...  False  False  False  False  False  False  False  False
18933  False  ...  False  False  False  False  False  False  False  False
18934  False  ...  False  False  False  False  False  False  False  False
18935  False  ...  False  False  False  False  False  False  False  False

          z1  flags_r
0      False    False
1      False    False
2      False    False
3      False    False
4      False    False
...      ...      ...
18931  False    False
18932  False    False
18933  False    False
18934  False    False
18935  False    False
```

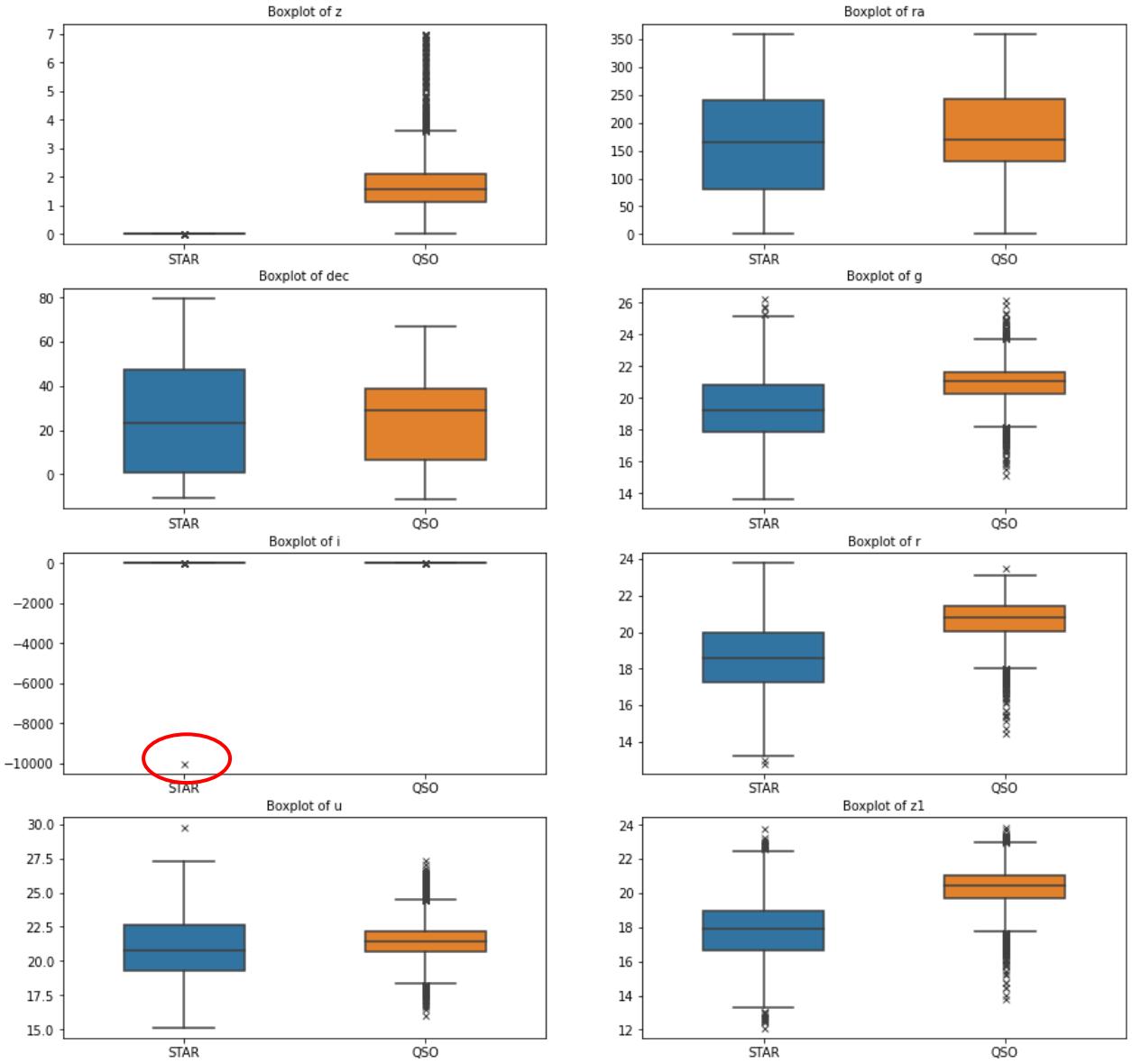
Check for any null values

STAR	z	ra	dec	g	i	r	u	z1
count	9389.000000	9389.000000	9389.000000	9389.000000	9389.000000	9389.000000	9389.000000	9389.000000
mean	-0.000100	167.137165	25.958400	19.321704	17.055428	18.543414	20.967977	17.883356
std	0.000430	110.379959	25.878316	1.978399	103.393383	1.799167	2.245343	1.722952
min	-0.004136	0.024207	-10.653739	13.674870	-9999.000000	12.818640	15.113150	12.113370
25%	-0.000262	80.114679	0.636428	17.897130	16.853150	17.232250	19.313230	16.634360
50%	-0.000073	164.079441	23.120849	19.247400	18.197550	18.584290	20.754860	17.911040
75%	0.000075	241.369002	47.408254	20.828670	19.278420	19.940950	22.626480	18.984500
max	0.004153	359.968600	79.697191	26.293190	24.361830	23.792090	29.766760	23.757610

QSO	z	ra	dec	g	i	r	u	z1
count	9547.000000	9547.000000	9547.000000	9547.000000	9547.000000	9547.000000	9547.000000	9547.000000
mean	1.674468	179.514638	25.576769	20.916066	20.448110	20.633022	21.458653	20.296976
std	0.832151	89.691032	17.835114	1.089440	1.022211	1.021341	1.380856	1.038817
min	0.000461	1.218573	-11.075809	15.144840	14.077170	14.438190	15.994070	13.823710
25%	1.110024	130.786298	6.471214	20.283695	19.848150	20.034980	20.633935	19.694085
50%	1.582601	169.976030	29.041854	21.062590	20.599810	20.798230	21.465280	20.414820
75%	2.119903	242.257677	38.616893	21.673040	21.193970	21.394330	22.176550	21.009405
max	7.015643	359.809734	66.718622	26.202820	23.578450	23.485970	27.378480	23.846600

Glimpse of an error data in the ‘i’ band.

Data Selection and pre-processing : EDA



The outliers in ‘i’ band is clearly visible, i have to discard this data

```
[ ]  display(df[df['i'] == min(df['i'])][main_cols + ['class']])
```

	z	ra	dec	g	i	r	u	z1	class
12087	-0.000383	334.209899	70.03263	16.24309	-9999.0	15.09408	18.71041	14.26493	STAR

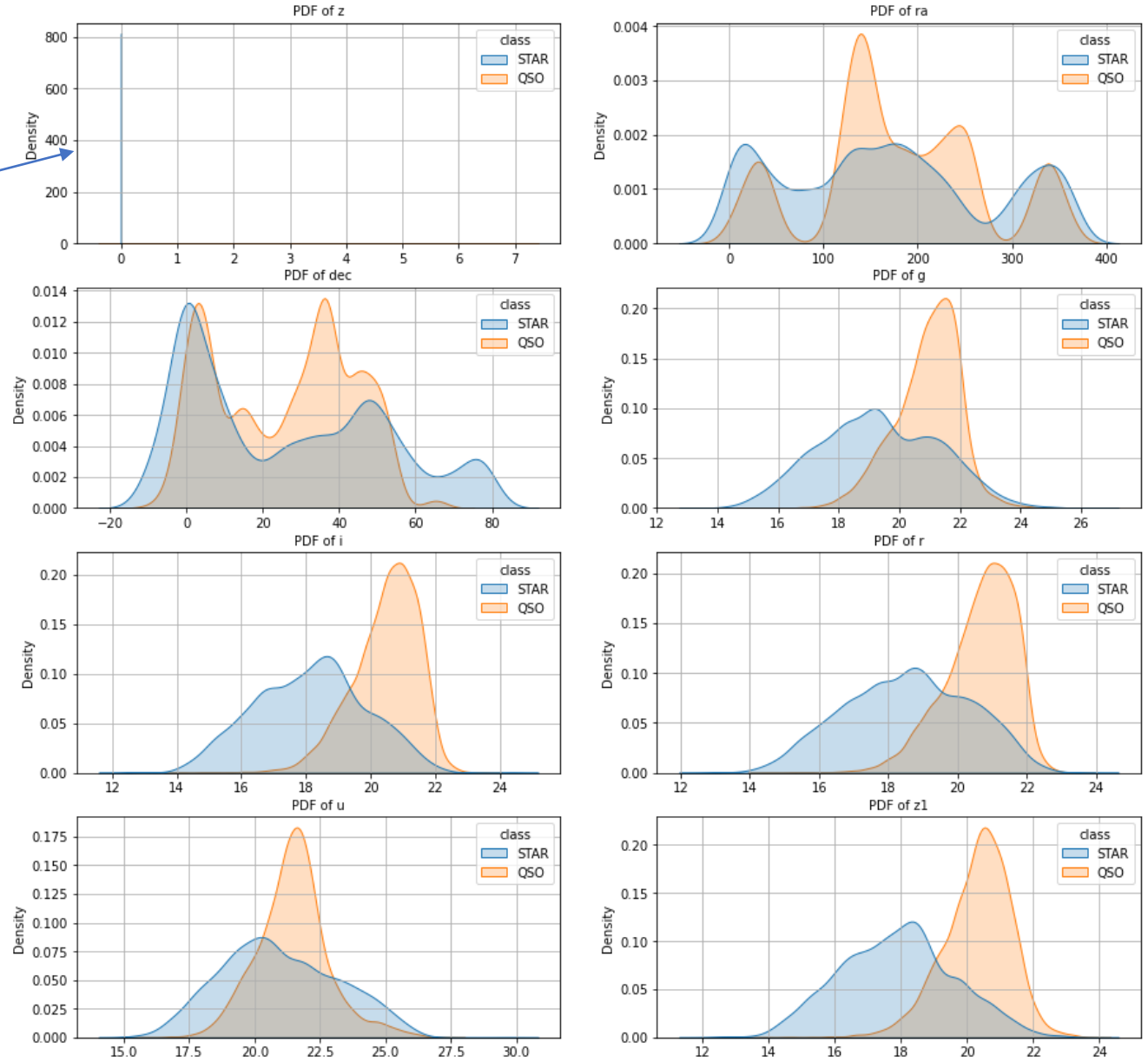
```
[ ]  index= df[df['i'] == min(df['i'])].index.to_list()[0]
```

```
[ ]  df = df.drop(index=[index])
     box_plotter(df=df, features=main_cols, target='class')
```

I identify the index and through the drop function discard this data.

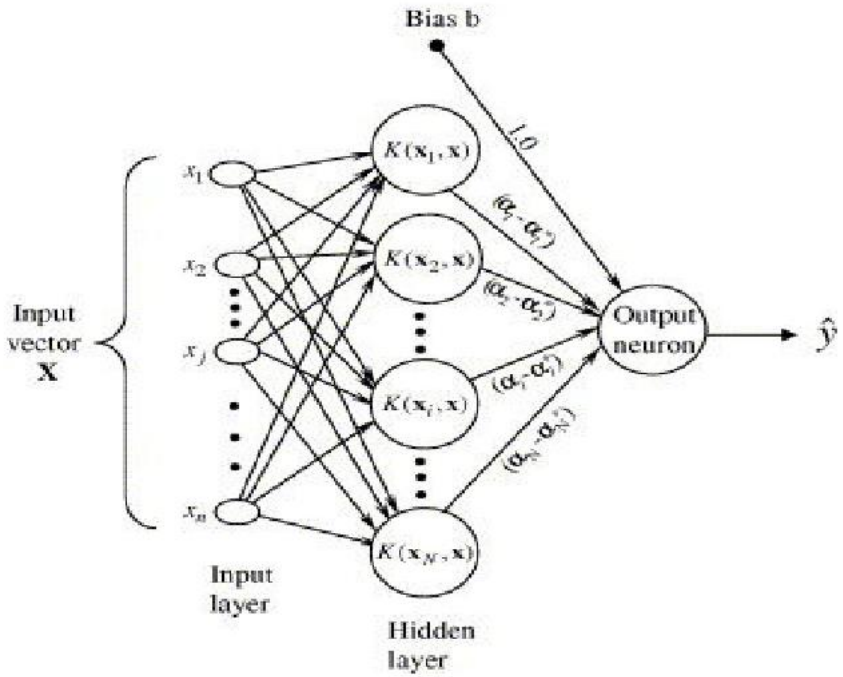
Data Selection and pre-processing : EDA

We can see an overlap of the plots of ra and dec, and a clearly distinction in the redshift's one. While, for the magnitude, the distribution peak is clearly different, shifted at higher value for qso than star.



A PDF could give us an hint on how the data are distribuited in the dataframes.

Support Vector Machine: Model Definition



SVM takes as inputs the photometrical parameters of DataFrame, apply a Kernel (in this case RBF) and gives as output the class.

The aim is:

Support Vector Machine: Model Definition

Input Data Rescaling: optimize the alghoritm and avoid attributes in greater numeric ranges dominating those in smaller numeric ranges.

```
array([[ 0.71182359, -0.82326349,  0.        , ...,  0.00865468,
         0.71513998, -0.14389582],
       [-0.53608593,  1.31208973,  0.        , ...,  0.15270903,
         0.87779062, -0.14389582],
       [ 0.28072757,  0.9501551 ,  0.        , ..., -0.92063971,
        -0.02834538, -0.14389582],
       ...,
       [-0.10498992,  3.33000942,  0.        , ...,  1.44398709,
         0.39732759, -0.14389582],
       [ 2.18662575,  0.33751167,  0.        , ...,  0.25384112,
         0.83453925, -0.14389582],
       [-1.26592392, -0.82297368,  0.        , ...,  1.73922114,
         0.35175134,  7.00868982]])
```

Through sklearn i transform the output class, that are labels, in a vector of 1 and 0 variables.

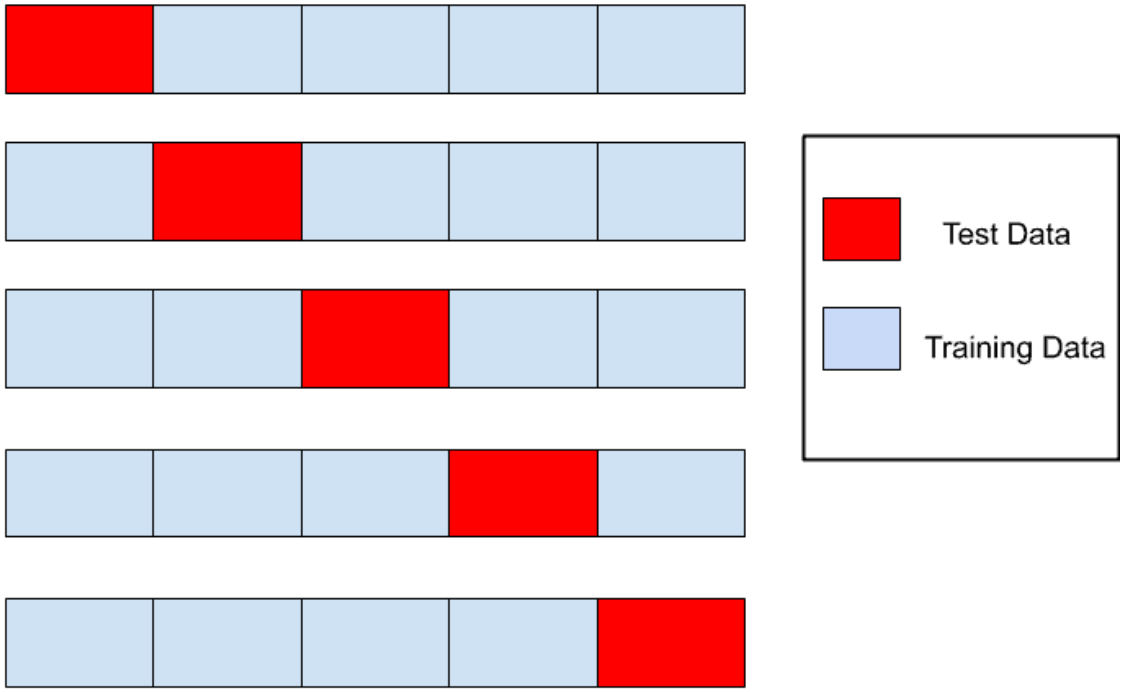
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

```
classi = list(le.classes_)
```

DATA SPLIT :

- Training Set : 70%
- Test set: 30%

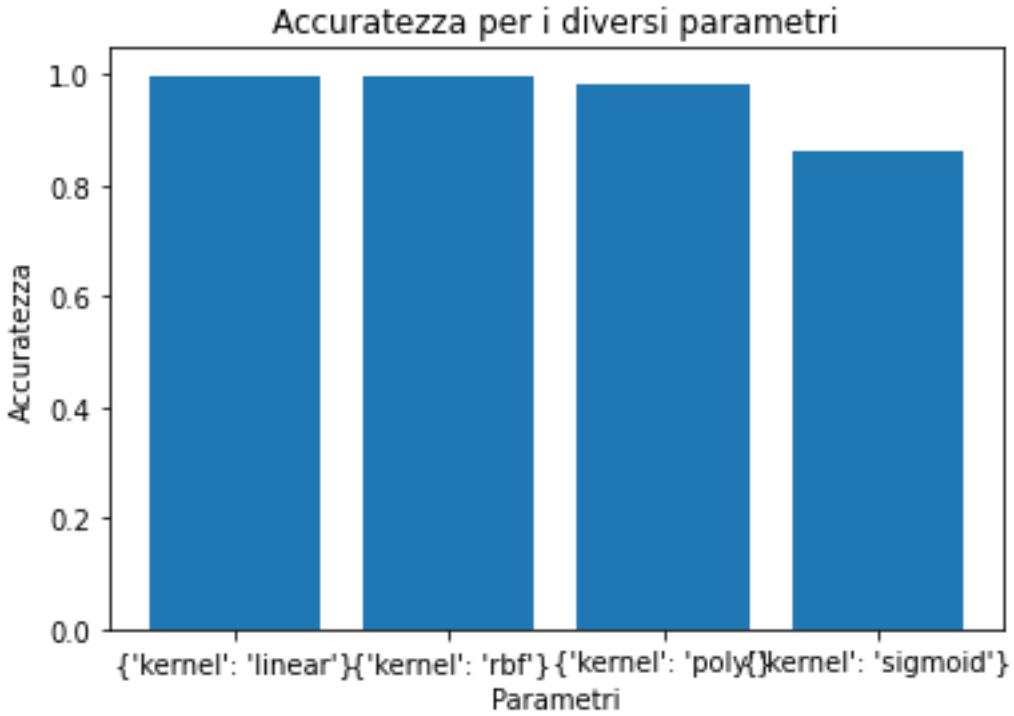
Support Vector Machine: Model Definition



K-fold cross-validation to evaluate the accuracy of the model

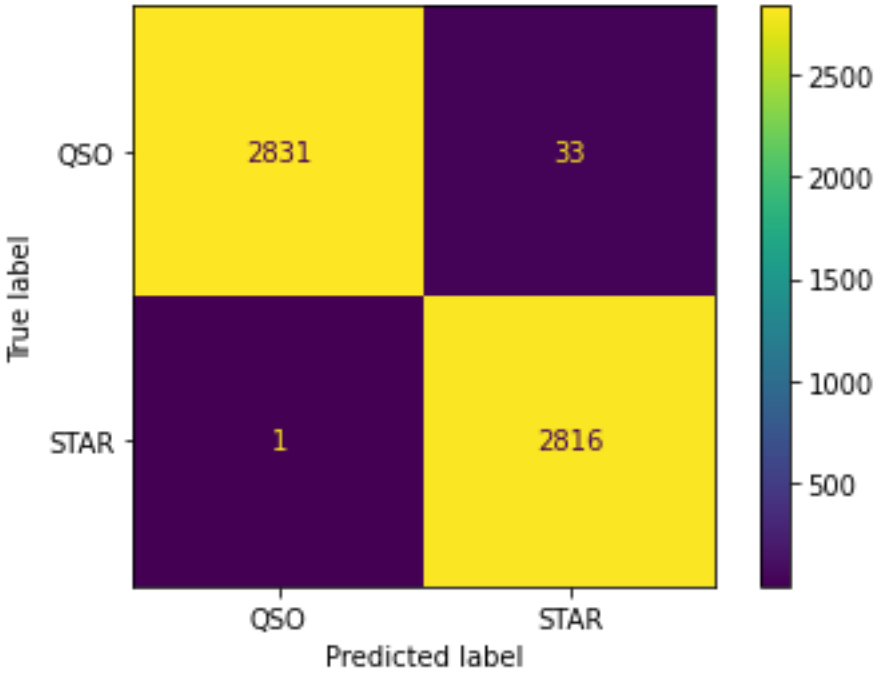
```
Accuratezza: 0.994 (+/- 0.004)
```

GridSearch identify linear and rbf kernel as the best kernel for this model



Performance and Conclusion: Confusion Matrixs

Confusion Matrixs (CM) are useful to represent the summary of the prediction results.



$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 0.999$$

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} = 3.5 \times 10^{-4}$$

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 0.988$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 0.011$$

TN	FP
FN	TP

Accuracy: 99.4% on test set

Performance and Conclusion : Usefull Statistical Parameter

➢ Other metrics:

- **Precision :**

Percentage of TP out of all the positive predicted

$$Precision = \frac{TP}{TP + FP}$$

- **Recall :**

Percentage of positive predicted out of the total positive (it is the same of TPR)

$$Recall = \frac{TP}{TP + FN}$$

- **F1-score :**

Armonic mean of precision and recall

$$F1 - Score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 * (Precision * Recall)}{Precision + Recall}$$

	precision	recall	f1-score	support
QSO	1.000	0.988	0.994	2864
STAR	0.988	1.000	0.994	2817
accuracy			0.994	5681
macro avg	0.994	0.994	0.994	5681
weighted avg	0.994	0.994	0.994	5681

Performance and Conclusion : Conclusion

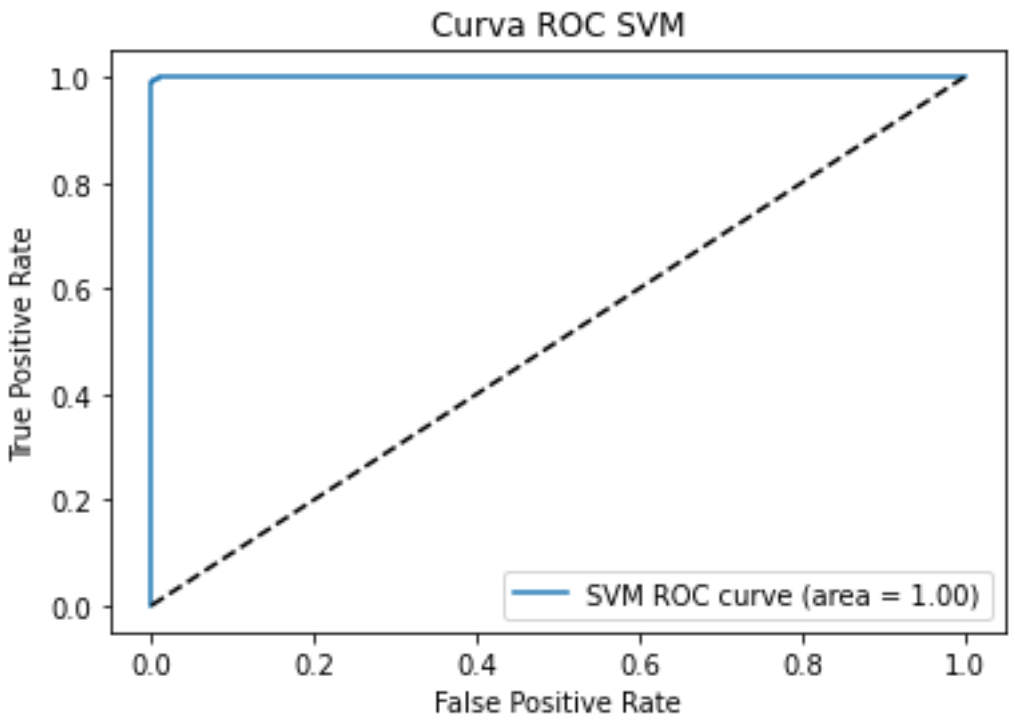
- In conclusion, thanks to the pre-processing of datas, the model has a high level of reliability in the classification of QSOs and STARS

- The model has a high capacity to create a pure cluster of QSO and it's seems that this sample is, also, complete (Precision and recall).

	precision	recall
QSO	1.000	0.988
STAR	0.988	1.000

- This high value could hint a problem in the model is not necessary a good thing, so we double check the reliability through F1- Score and ROC Curve

Performance and Conclusion : Conclusion



The AUC of the ROC curve tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing the classes

Is the SVM the only model that could give us good results?

- ***Random Forest:***

  It's based on decision trees, combining them for classification. The trees are builded using a random selection of variables and training data. The final model makes a prediction based on the average predictions of all the trees

- ***Neural Network:***

  Model based on series of layers of interconnected neurons to analyze input data and provide an output prediction. There are several architecture of Neural Network used for binary classification, among them there are CNN and RNN. CNN are prefered for processing data as images, where the information is organized into a grid of pixels. RNN are ideal for processing sequential data, such as sound or text, where the order of the information is relevant.

- ***K-Nearest Neighbors (KNN):***

  KNN is a model that uses a distance measure to find the k closest training sample to the input point and use the majority of these training examples to predict the class of input data.

THE END