CLASSIFICATION OF RADAR RETURNS FROM THE IONOSPHERE USING NEURAL NETWORKS AND MACHINE LEARNING TECHNIQUES

Gruppo 2

Valentina Civita (894036)
William Colombini (892235)
Ilaria Molinari (905443)

Problem

The problem we are facing is to classify a series of data returns from a set of radars that recognize patterns in the ionosphere, so that they will be available for further analysis. The dataset analyzed is an open source dataset and it has been donated by V. Sigillito [https://archive.ics.uci.edu/ml/datasets/Ionosphere].

We have available 351 samples, each one is described by 34 features, that represents the Real and the Imaginary part of the spectra.

Our sample are classified as “good” (G1) or “bad” (G0) samples, meaning the good ones are suitable for further analysis, while the bad ones are not and need to be discarded.

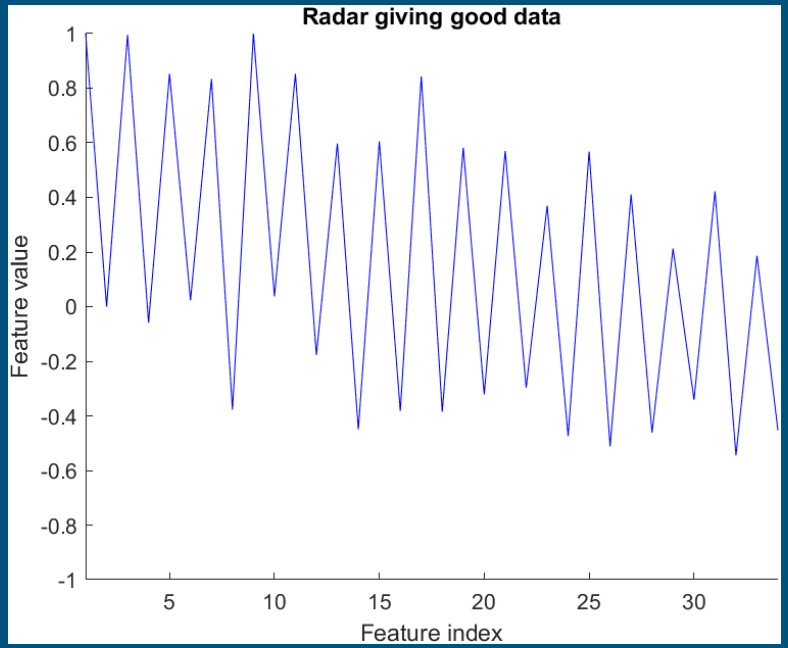
Problem

We present a couple of examples to show that they are not so easily distinguishable by a human eye.

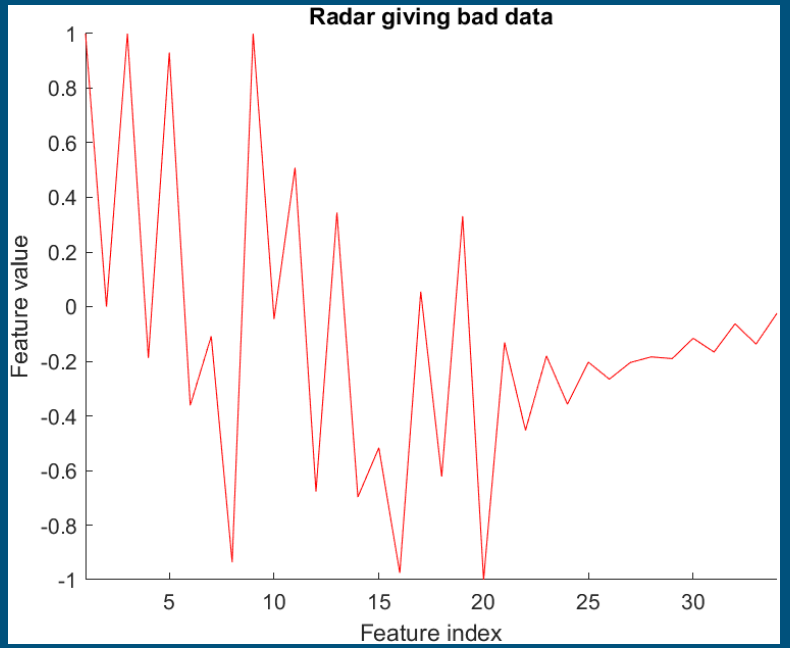
So we implemented different machine learning techniques to separate the “good” (G1) ones from the “bad” (G0) ones.

Furthermore our methods might be helpful in case of a large amount of spectra to sort, since doing that “by hand” could be an impossible task.

Problem



Examples of good data returns (G1)



Examples of bad data returns (G0)

Problem

This data set had already been analyzed using single layer and multiple layers neural networks [1].

Our goal is to try to better the performances obtained in this publication by using single layer neural networks combined with machine learning algorithms.

[1] V. G. Sigillito, S. P. Wing, L. V. Hutton, K. B. Baker, 1989, *Classification of radar returns from the ionosphere using neural networks.*

Problem

VINCENT G. SIGILLITO, SIMON P. WING, LARRIE V. HUTTON, and KILE B. BAKER

CLASSIFICATION OF RADAR RETURNS FROM THE IONOSPHERE USING NEURAL NETWORKS

In ionospheric research, we must classify radar returns from the ionosphere as either suitable for further analysis or not. This time-consuming task has typically required human intervention. We tested several different feedforward neural networks to investigate the effects of network type (single-layer versus multilayer) and number of hidden nodes upon performance. As expected, the multilayer feedforward networks (MLFN's) outperformed the single-layer networks, achieving 100% accuracy on the training set and up to 98% accuracy on the testing set. Comparable figures for the single-layer networks were 94.5% and 92%, respectively. When measures of sensitivity, specificity, and proportion of variance accounted for by the model are considered, the superiority of the MLFN's over the single-layer networks is even more striking.

[1] V. G. Sigillito, S. P. Wing, L. V. Hutton, K. B. Baker, 1989, *Classification of radar returns from the ionosphere using neural networks.*

Our approach

First of all we used a single layer neural network, in a similar way as described in the previous article, so that we can compare our results to the ones obtained in the article.

Secondly, we tried to approach this problem by implementing two machine learning algorithms: Random forest and SVM.

And lastly, in order to better the performances obtained, we implemented a multi decision maker to compare which algorithm works the best.

Single-Layer Neural Network

In order to compare our results to the ones in the article mentioned before, we decided to create a single-layer neural network using the very same training and testing data sets.

Therefore, we trained the networks with a training set of 200 returns (101 good, 99 bad) and we tested it with a testing set of 151 returns, of which 124 were good and 27 were bad.

Furthermore, we trained our algorithm 100 times, so that we were able to calculate the mean and standard deviation of the metrics values.

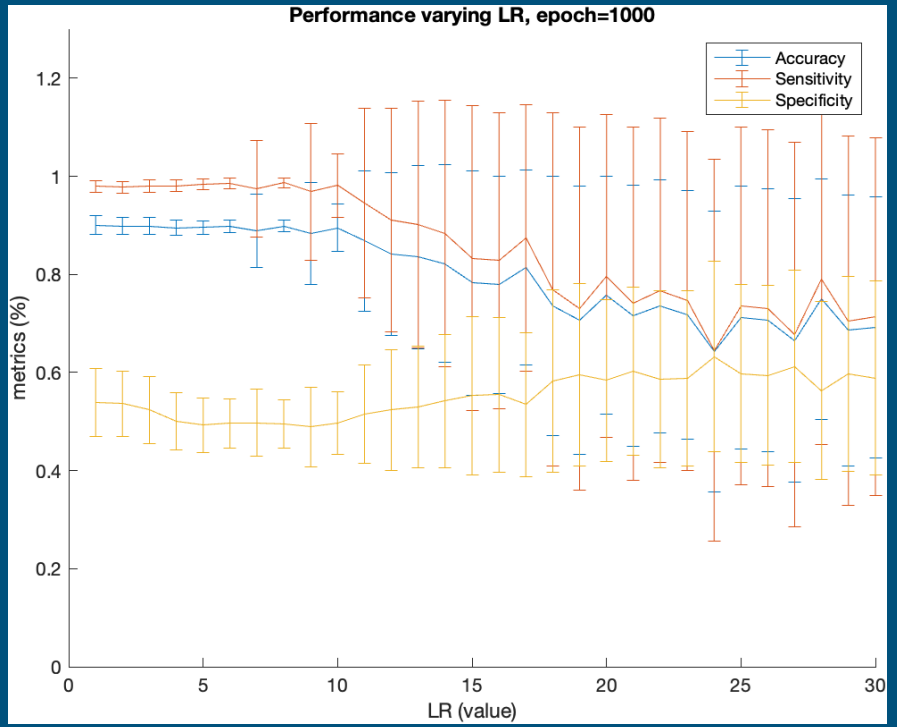
We need to select the parameters that give us the best performances with said setting.

Single-Layer Neural Network

We want to create a multi-decider to compare all our different methods, but to do so we need to find the parameters that give us the best performances.

In order to choose the best Learning Rate we changed it while maintaining fixed the number of epochs at 1000 :

- Between 1 and 30 ;

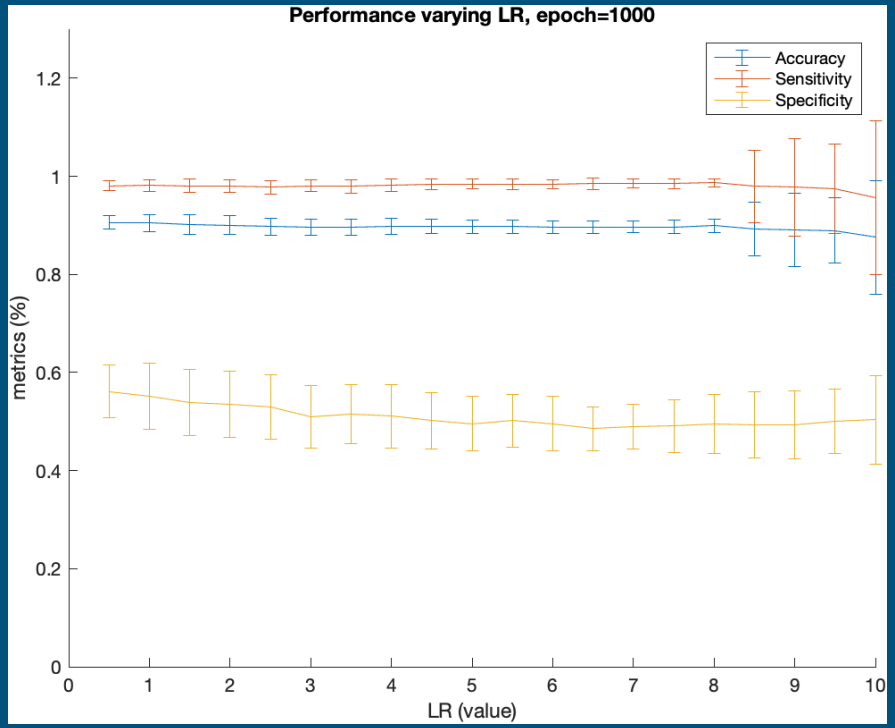


Single-Layer Neural Network

We want to create a multi-decider to compare all our different methods, but to do so we need to find the parameters that give us the best performances.

In order to choose the best Learning Rate we changed it while maintaining fixed the number of epochs at 1000 :

- Between 1 and 30 ;
- Between 0.5 and 10 ;

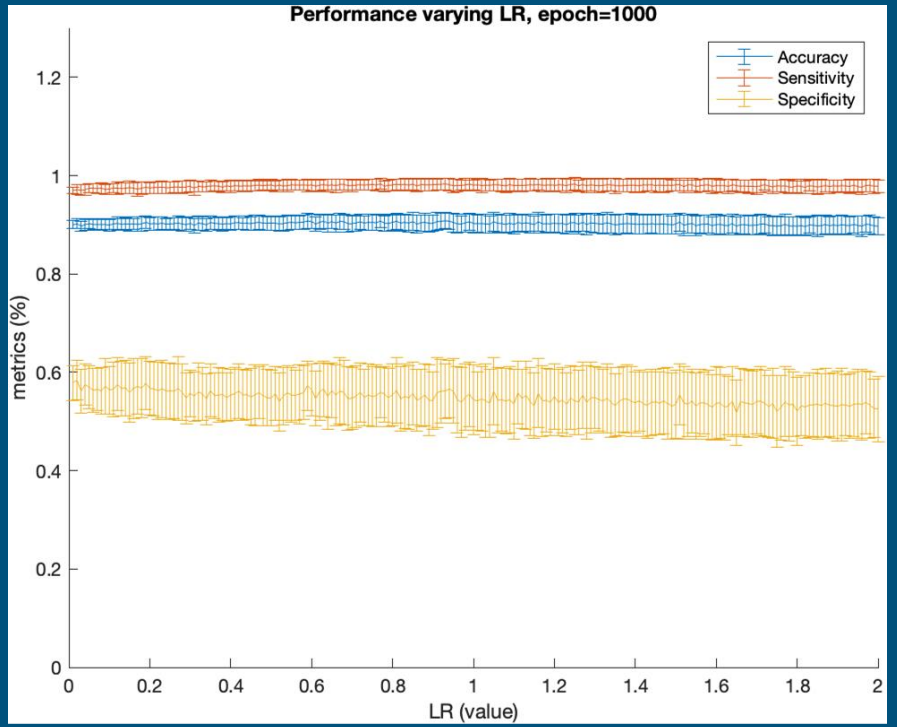


Single-Layer Neural Network

We want to create a multi-decider to compare all our different methods, but to do so we need to find the parameters that give us the best performances.

In order to choose the best Learning Rate we changed it while maintaining fixed the number of epochs at 1000 :

- Between 1 and 30 ;
- Between 0.5 and 10 ;
- Between 0.01 and 2 .



Neural Network: best performances

TPR mean	TPR std	TNR mean	TNR std	FPR mean	FPR std	FNR mean	FNR std	acc mean	acc std	LR
0,983468	0,011988	0,562222	0,055806	0,437778	0,055806	0,016532	0,011988	0,908146	0,016483	0,93
0,983468	0,011135	0,559630	0,051010	0,440370	0,051010	0,016532	0,011135	0,907682	0,014145	0,77
0,980403	0,012674	0,572963	0,049789	0,427037	0,049789	0,019597	0,012674	0,907550	0,015175	0,59
0,982258	0,010505	0,563704	0,066292	0,436296	0,066292	0,017742	0,010505	0,907417	0,016652	0,81
0,981694	0,010434	0,565926	0,056717	0,434074	0,056717	0,018306	0,010434	0,907351	0,014597	0,63
0,981613	0,012189	0,566296	0,058215	0,433704	0,058215	0,018387	0,012189	0,907351	0,016799	0,94

Neural Network: best performances

Overall the best performances obtained with this technique were those using a Learning Rate value of 0.93 and 1000 epochs.

In the confusion matrix beside are reported the values for the True Positives and True Negatives on the diagonal that allowed us to estimate the sensitivity and specificity percentage (shown at the bottom).

This data present an error due to the fact that every time it is trained 100 times, so we have a mean value and a standard deviation.

Neural Network (LR=0.93 EP=1000) - time ≈ 8.4 sec

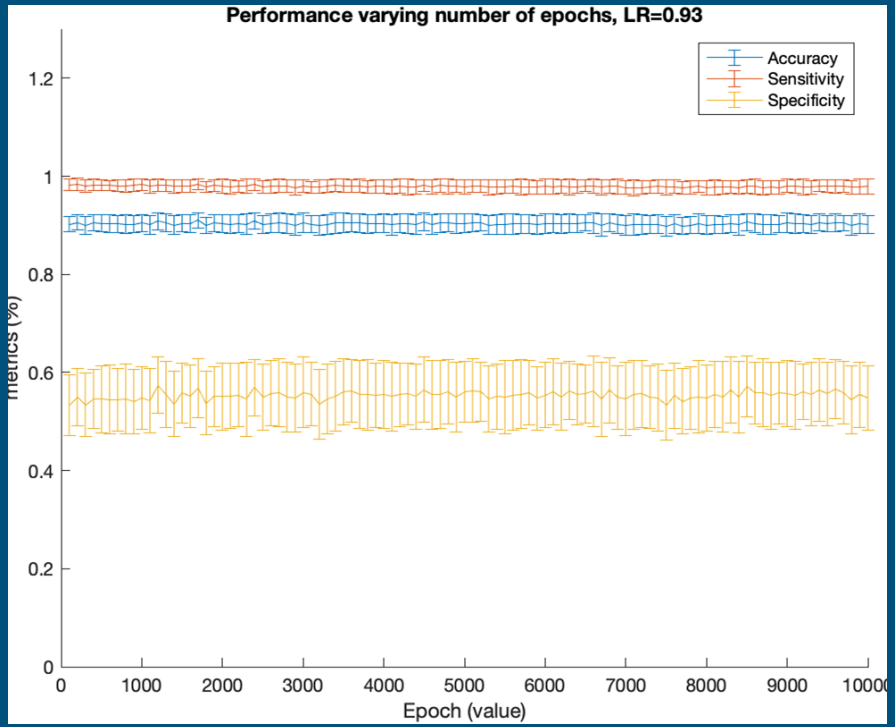
TOTAL SAMPLES 151	ACTUAL POSITIVE 124	ACTUAL NEGATIVE 27
PREDICTED POSITIVE 124 ± 4	TRUE POSITIVE 122 ± 4	FALSE POSITIVE 12 ± 2
PREDICTED NEGATIVE 17 ± 4	FALSE NEGATIVE 2 ± 2	TRUE NEGATIVE 15 ± 2
ACCURACY 90.81% ± 1.65%	SENSITIVITY 98.35% ± 1.2%	SPECIFICITY 56.22% ± 5.58%

Single-Layer Neural Network

A similar analysis was done fixing the chosen Learning Rate (0.93) and varying instead the number of epochs between 100 and 10000.

We did not find an important variation in the performances (as can be seen in the picture) and decided then to stick with 1000 epochs.

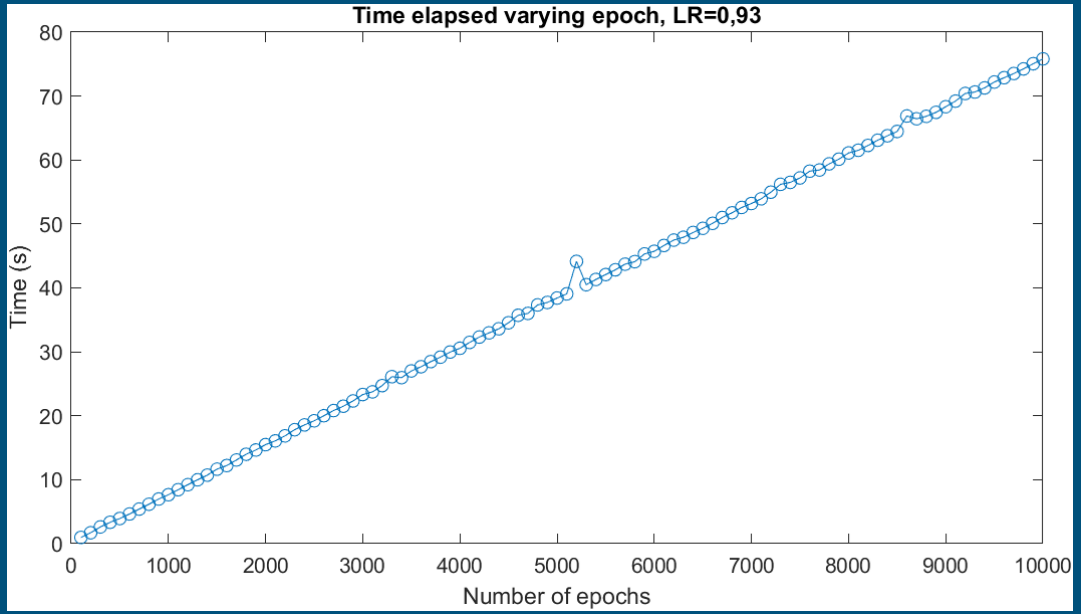
Therefore, in this particular case, the metrics values seen before still stand.



Single-Layer Neural Network

Our choice is furthermore justified by the time needed to train the algorithm for a high number of epochs.

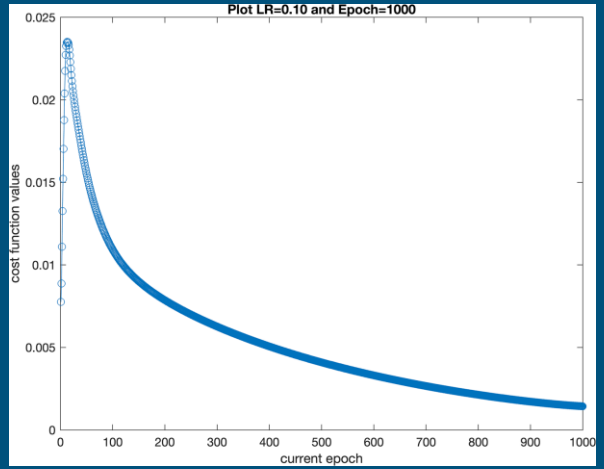
For training an algorithm with 1000 epochs for 100 cycles we need about 8.4 seconds, while to do the same with 10000 epochs we need about 75.8 seconds.

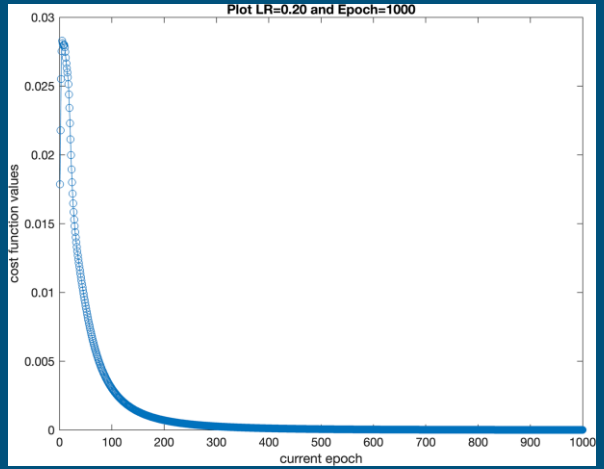


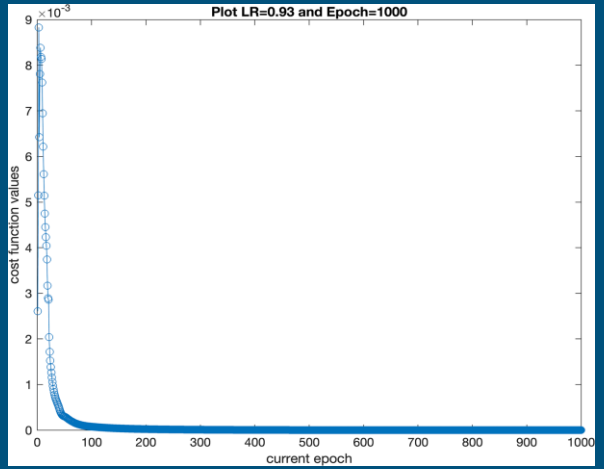
Single-Layer Neural Network

Is 1000 epochs enough? To be sure of that we plotted the value of the cost function (0.5(expected - predicted)^2) varying the number of epochs and for different learning rates.

As we can see from the pictures below, the cost function (which represents our error) does not reach zero in 1000 epochs if the LR is too low, but for LR=0.2 we can already say that 1000 epochs is enough for the cost function to be approximately zero. Since we chose LR=0.93, we are able to say that 1000 epochs is enough to value the weights correctly.







K-folding problem: unbalanced data

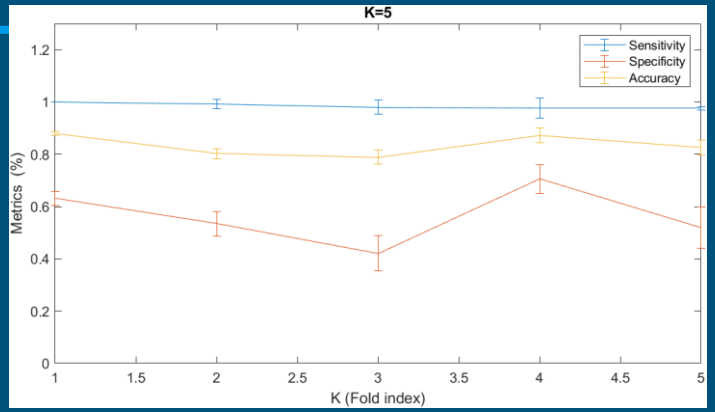
The main problem of this data set is that it is very unbalanced: of the 351 samples, **225** are **good (G1)** data returns and **126** are **bad (G0)**.

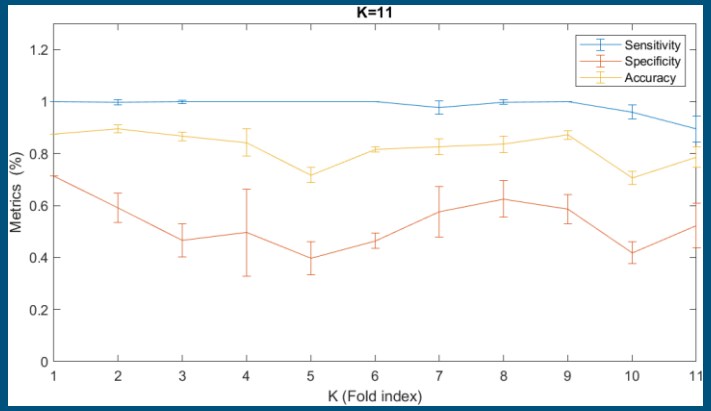
We tried to implement a K-folding algorithm.

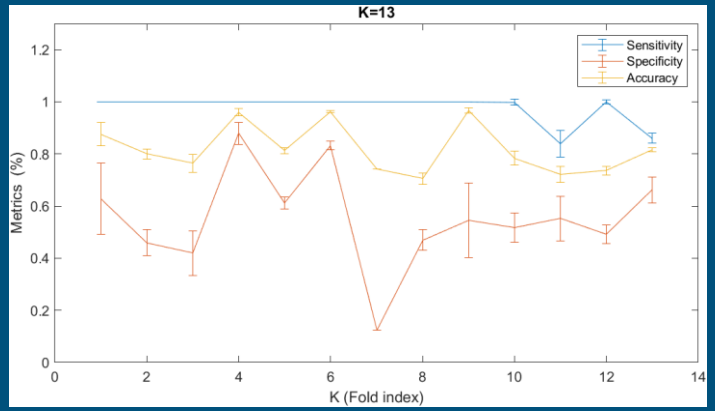
For this specific case it is difficult to write an algorithm that creates folds with a balanced number of good and bad data, so we made so that the folds were filled randomly, without taking into account if they were balanced or not.

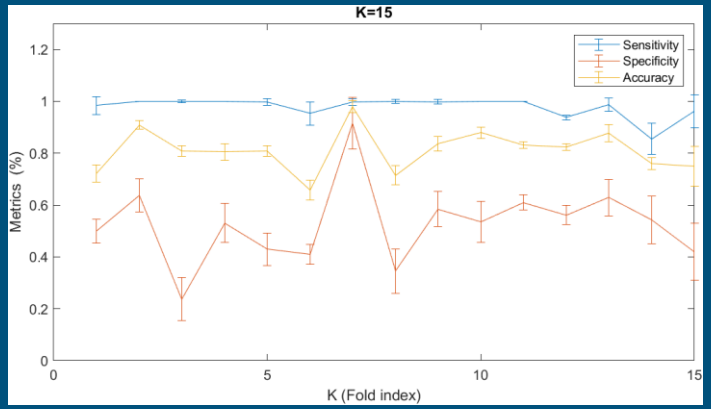
We expected that by increasing the number of folds, some of them would have a very low number of bad data, therefore we expected low performances on those folds. We tested this theory on the neural network algorithm.

K-folding problem: unbalanced data









K-folding problem: unbalanced data

We notice that for a higher number of folds the specificity significantly worsen, as expected.

We tried the same procedure with the SVM obtaining similar results, with performances that worsen when increasing the number of folds.

The random forest algorithm is particularly problematic because the time needed to compute only with K=3 was really high, more than one hour.

So we abandoned the idea of using a K-folding majority rule method and we searched for another method to select our training data.

Our solution: create a balanced training set

Now the problem is to select HOW MANY data to use for the training, without taking every time the same samples.

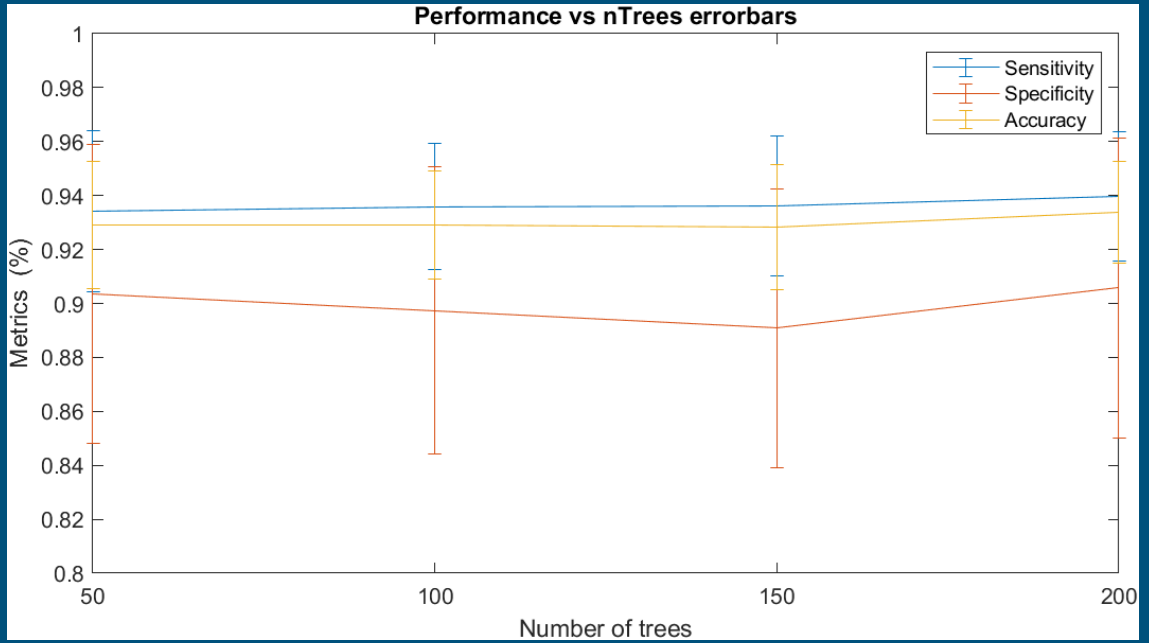
Following the same procedure described in the cited article, we decided to select the data in a more balanced way.

We chose to train on an even number of data, divided into half good and half bad. These samples are chosen randomly every time. The remaining samples are used for the internal testing.

Random Forest: number of trees

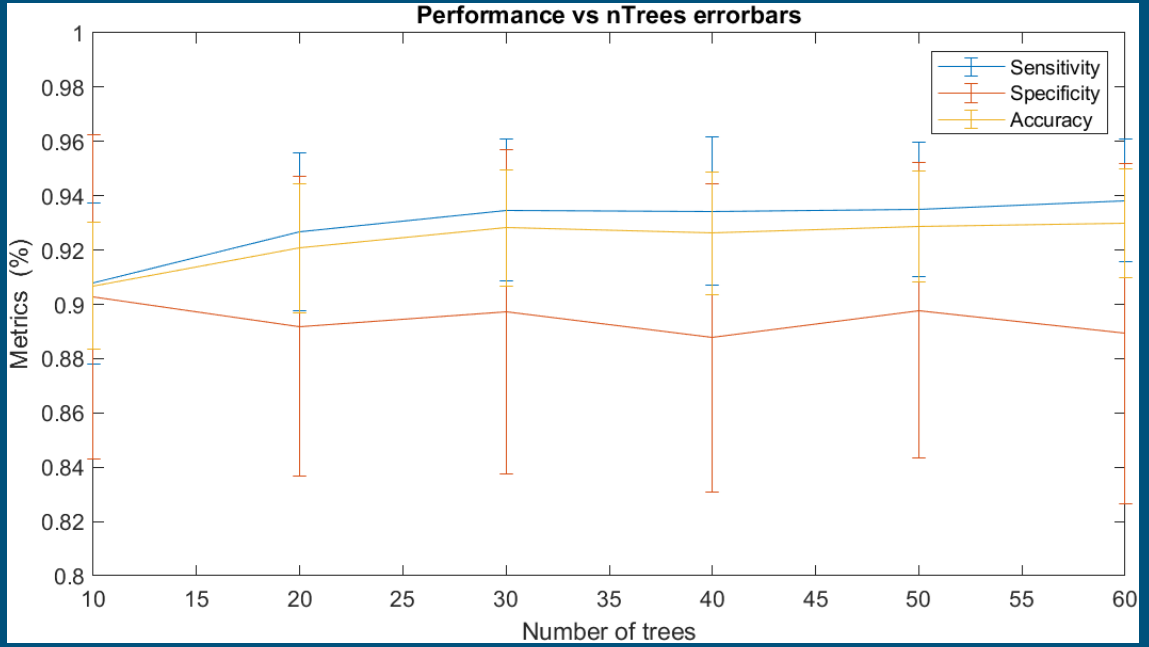
Found the best number of data to train the algorithm we had to choose the best number of trees.

Since this is the most time consuming algorithm, we began by changing the number of trees by 50 at a time.



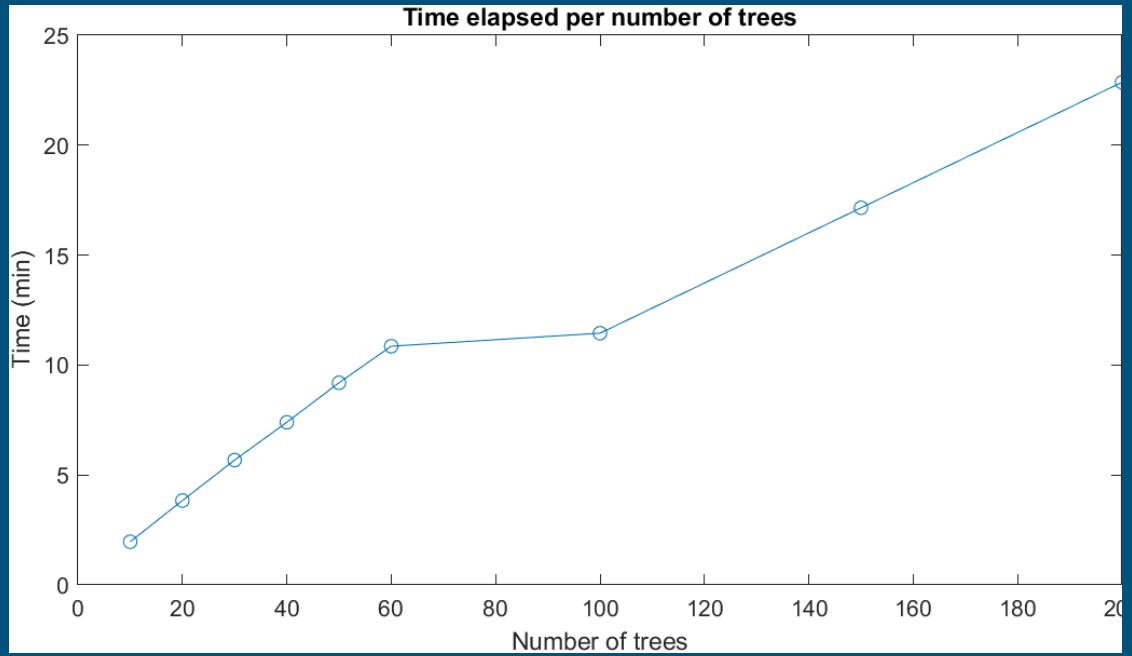
Random Forest: number of trees

The performances and the errors indicate that this values do not vary so much, so we opted for a smaller number of trees. We ran the code by changing the number of trees with a finer grain, by 10 at a time.



Random Forest: number of trees

Our choice to keep a lower number of trees is justified by the time needed to train the algorithm for different number of trees.



Random Forest: best performances

The performances obtained with this algorithm are the following.

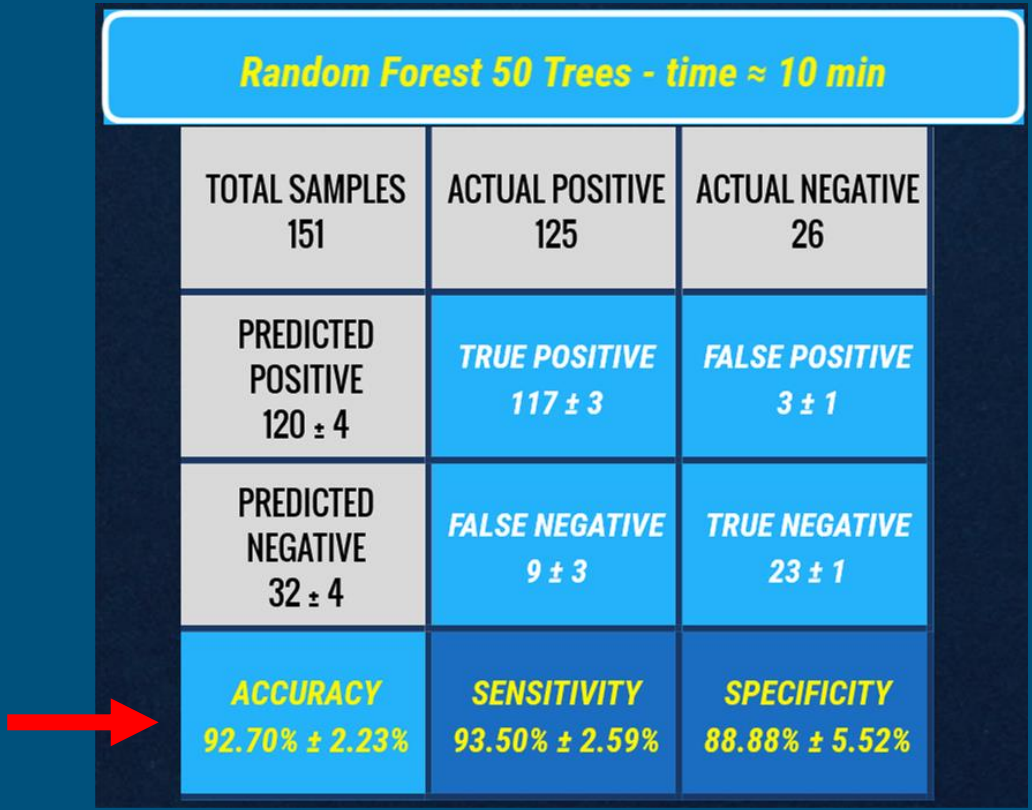
* From a KS test they result equivalent

TPR mean	TPR std	TNR mean	TNR std	FPR mean	FPR std	FNR mean	FNR std	acc mean	acc std	n trees
0,9382	0,0226	0,8892	0,0626	0,1108	0,0626	0,0618	0,0226	0,9298	0,0201	60
0,9350	0,0247	0,8977	0,0545	0,1023	0,0545	0,0650	0,0247	0,9285	0,0204	50
0,9346	0,0261	0,8973	0,0596	0,1027	0,0596	0,0654	0,0261	0,9282	0,0214	30
0,9343	0,0272	0,8877	0,0567	0,1123	0,0567	0,0657	0,0272	0,9263	0,0226	40
0,9268	0,0290	0,8919	0,0551	0,1081	0,0551	0,0732	0,0290	0,9208	0,0238	20
0,9077	0,0297	0,9027	0,0597	0,0973	0,0597	0,0923	0,0297	0,9068	0,0234	10

Random Forest: best performances

So the best performances are represented in this confusion matrix, where are shown the values for the True Positives and the True Negatives on the diagonal of the matrix, and on the bottom there are reported the values for sensitivity, specificity and accuracy.

This data present an error due to the fact that every time it is trained 100 times, so we have a mean value and a standard deviation.

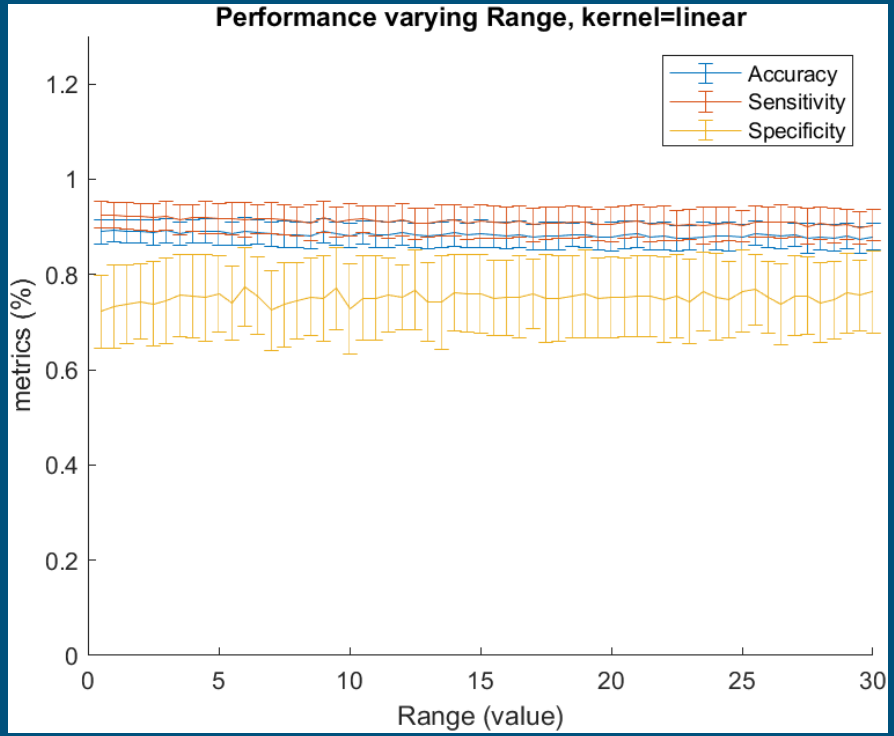


SVM with balanced train data

We made some runs with our custom “fold” data selection using SVM algorithm varying our hyperparameters.

So we decided to change the range value between 0.5 and 30 for different kernels:

- Linear Kernel ;

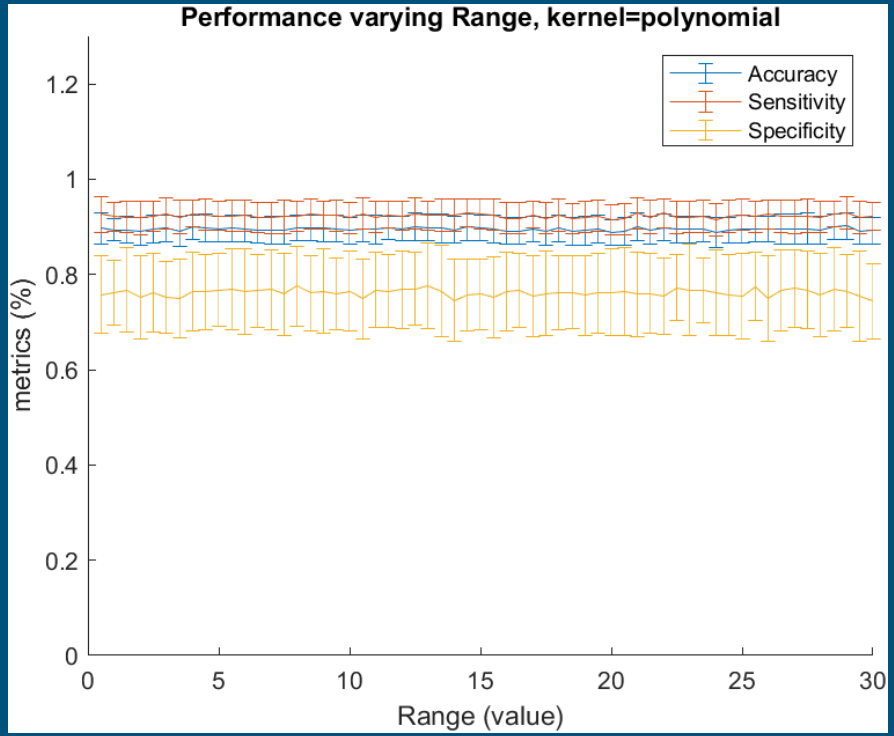


SVM with balanced train data

We made some runs with our custom “fold” data selection using SVM algorithm varying our hyperparameters.

So we decided to change the range value between 0.5 and 30 for different kernels:

- Linear Kernel ;
- Polynomial Kernel ;

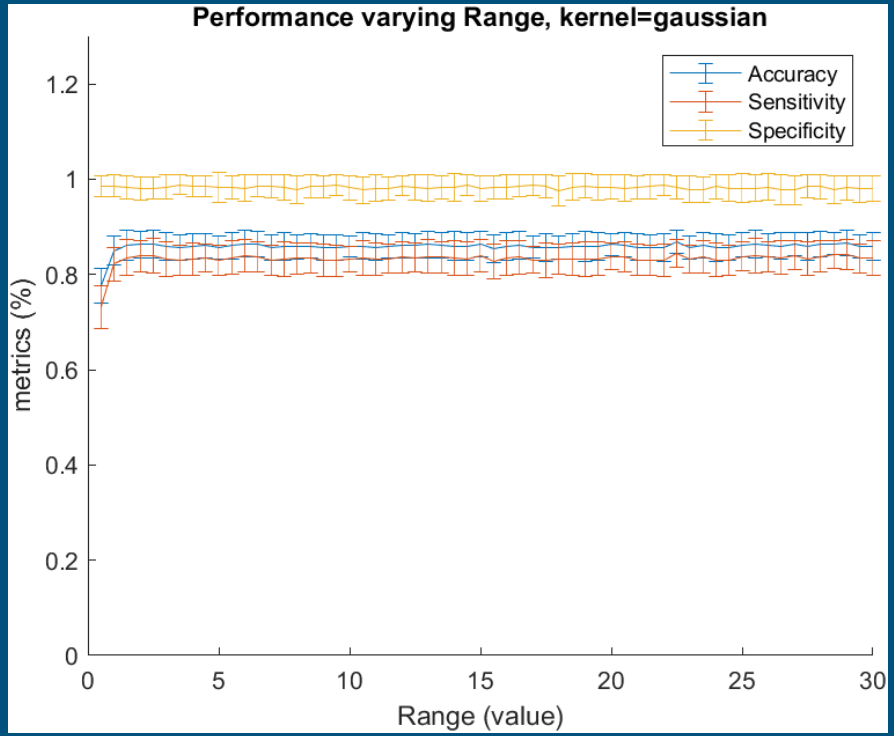


SVM with balanced train data

We made some runs with our custom “fold” data selection using SVM algorithm varying our hyperparameters.

So we decided to change the range value between 0.5 and 30 for different kernels:

- Linear Kernel ;
- Polynomial Kernel ;
- Gaussian Kernel .

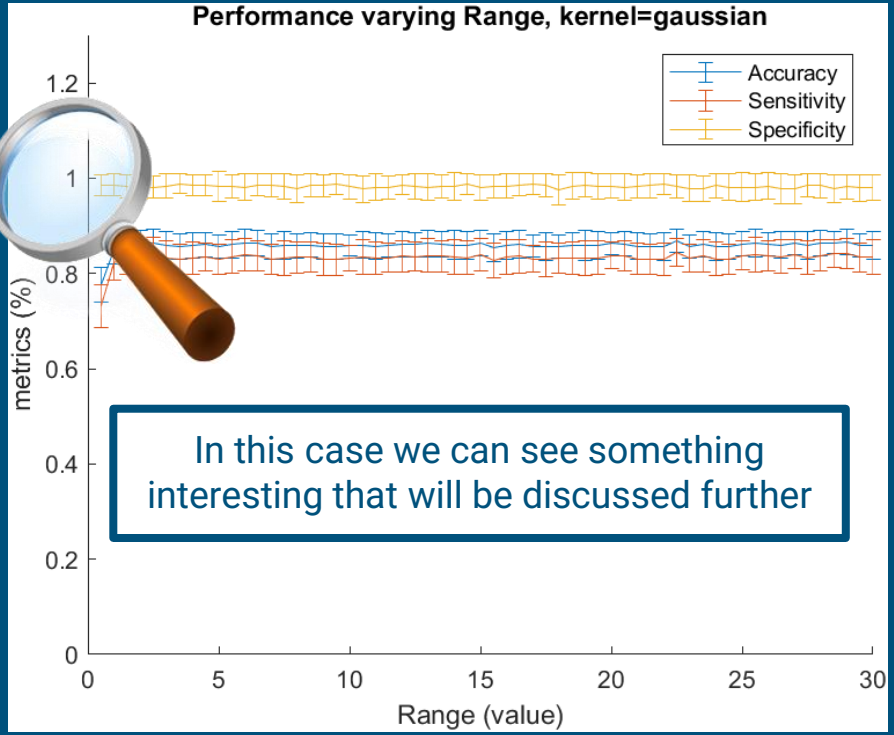


SVM with balanced train data

We made some runs with our custom “fold” data selection using SVM algorithm varying our hyperparameters.

So we decided to change the range value between 0.5 and 30 for different kernels:

- Linear Kernel ;
- Polynomial Kernel ;
- Gaussian Kernel .

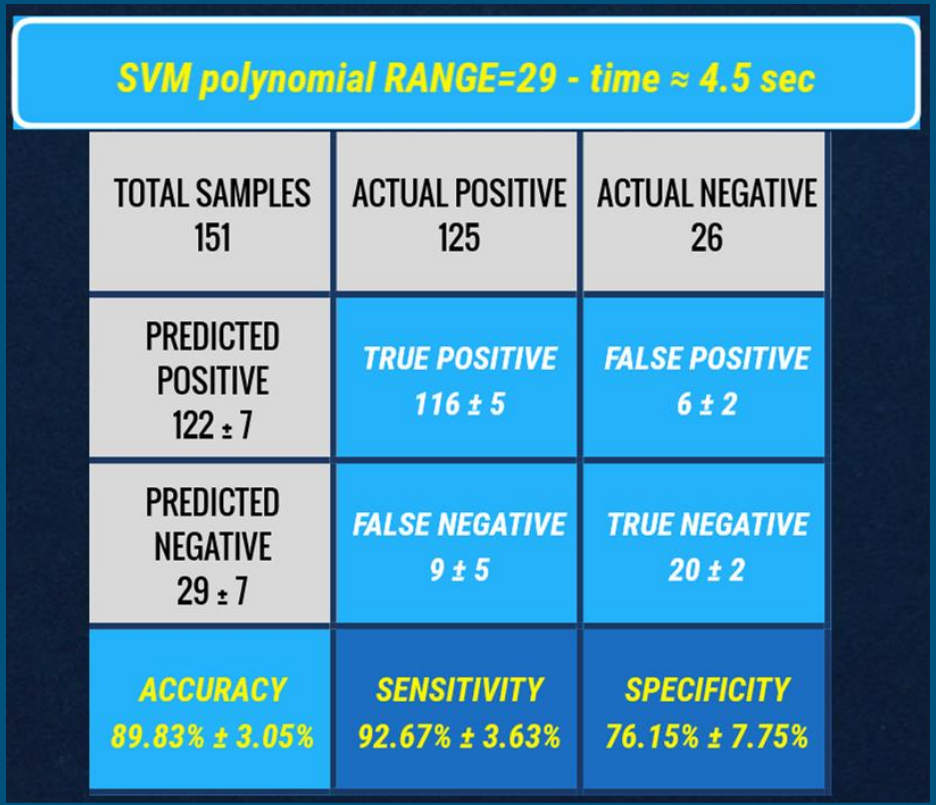


SVM balanced train data: best performances

	TPR mean	TPR std	TNR mean	TNR std	FPR mean	FPR std	FNR mean	FNR std	acc mean	acc std	Range
LINEAR	0,923280	0,030913	0,745000	0,090102	0,255000	0,090102	0,076720	0,030913	0,892583	0,025719	3,0
	0,925200	0,026359	0,732308	0,087794	0,267692	0,087794	0,074800	0,026359	0,891987	0,022800	1,0
POLYNOMIAL	0,929840	0,033728	0,765385	0,076044	0,234615	0,076044	0,070160	0,033728	0,901523	0,028405	29,0
	0,927920	0,033258	0,769615	0,076239	0,230385	0,076239	0,072080	0,033258	0,900662	0,028442	12,5
GAUSSIAN	0,842720	0,029415	0,980769	0,025931	0,019231	0,025931	0,157280	0,029415	0,866490	0,023919	12,5
	0,841840	0,035063	0,985000	0,024383	0,015000	0,024383	0,158160	0,035063	0,866490	0,029402	2,5

SVM balanced train data: best performances

As previously done we can see the confusion matrix for the best SVM code.



SVM balanced train data: best performances

As previously done we can see the confusion matrix for the best SVM code.

Random Forest 50 Trees - time ≈ 10 min

From a quick comparison with the RF method we can see that there is an important improvement in time but the performances aren't as good as before:

ACCURACY	SENSITIVITY	SPECIFICITY
92.70% ± 2.23%	93.50% ± 2.59%	88.88% ± 5.52%

SVM polynomial RANGE=29 - time ≈ 4.5 sec

TOTAL SAMPLES 151	ACTUAL POSITIVE 125	ACTUAL NEGATIVE 26
PREDICTED POSITIVE 122 ± 7	TRUE POSITIVE 116 ± 5	FALSE POSITIVE 6 ± 2
PREDICTED NEGATIVE 29 ± 7	FALSE NEGATIVE 9 ± 5	TRUE NEGATIVE 20 ± 2
ACCURACY 89.83% ± 3.05%	SENSITIVITY 92.67% ± 3.63%	SPECIFICITY 76.15% ± 7.75%

Principal Component Analysis

We tried at this point to find the most representative features of our dataset by performing a PCA.

1. Features were labeled for describing their physical meaning [1];

Re1	**Im1**	**Re2**	**Im2**	**...**	**Re16**	**Im16**	**Re17**	**Im17**

[1] V. G. Sigillito, S. P. Wing, L. V. Hutton, K. B. Baker, 1989, *Classification of radar returns from the ionosphere using neural networks.*

Principal Component Analysis

We tried at this point to find the most representative features of our dataset by performing a PCA.

1. Features were labeled for describing their physical meaning [1];
2. PCs were found, with relative eigenvalues;

[1] V. G. Sigillito, S. P. Wing, L. V. Hutton, K. B. Baker, 1989, *Classification of radar returns from the ionosphere using neural networks.*

Principal Component Analysis

We tried at this point to find the most representative features of our dataset by performing a PCA.

1. Features were labeled for describing their physical meaning [1];
2. PCs were found, with relative eigenvalues;
3. It was chosen a number of PCs using a graphical representation;

[1] V. G. Sigillito, S. P. Wing, L. V. Hutton, K. B. Baker, 1989, *Classification of radar returns from the ionosphere using neural networks.*

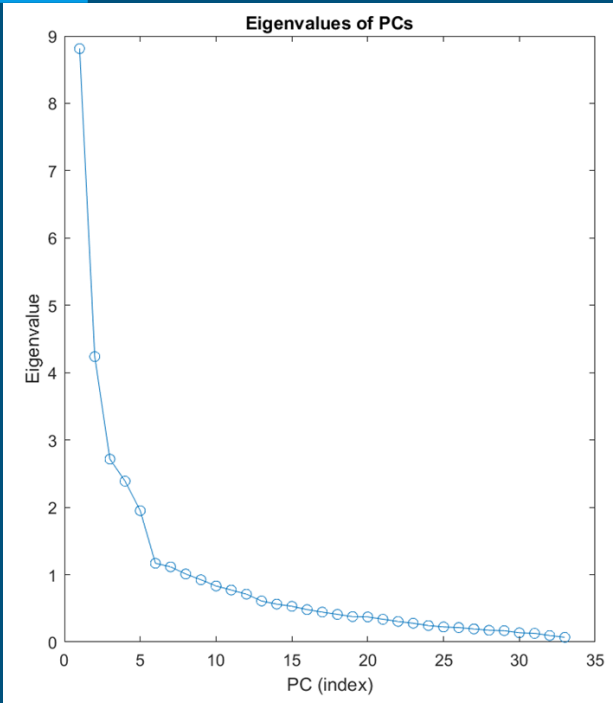
Principal Component Analysis

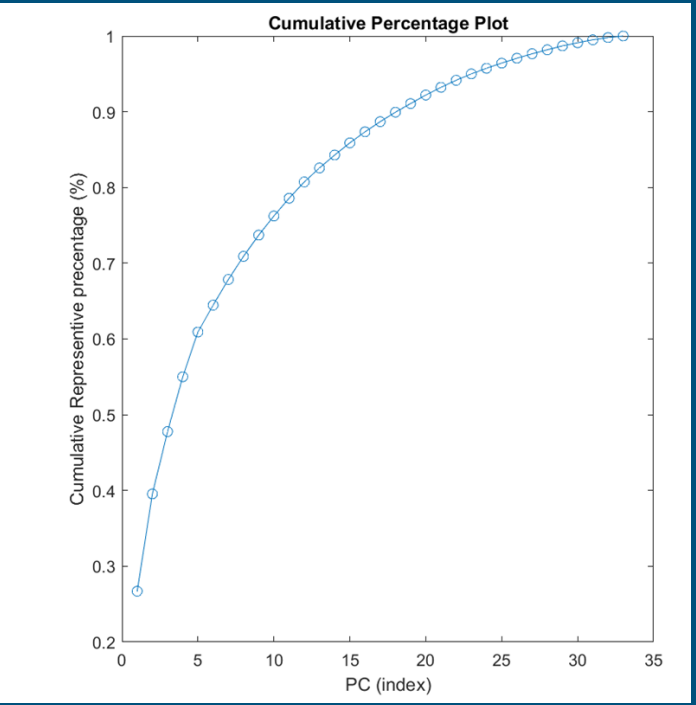
We tried at this point to find the most representative features of our dataset by performing a PCA.

1. Features were labeled for describing their physical meaning [1];
2. PCs were found, with relative eigenvalues;
3. It was chosen a number of PCs using a graphical representation;
4. Using a weighted average score we found the most significative features, which allowed us to go back to the features' space;

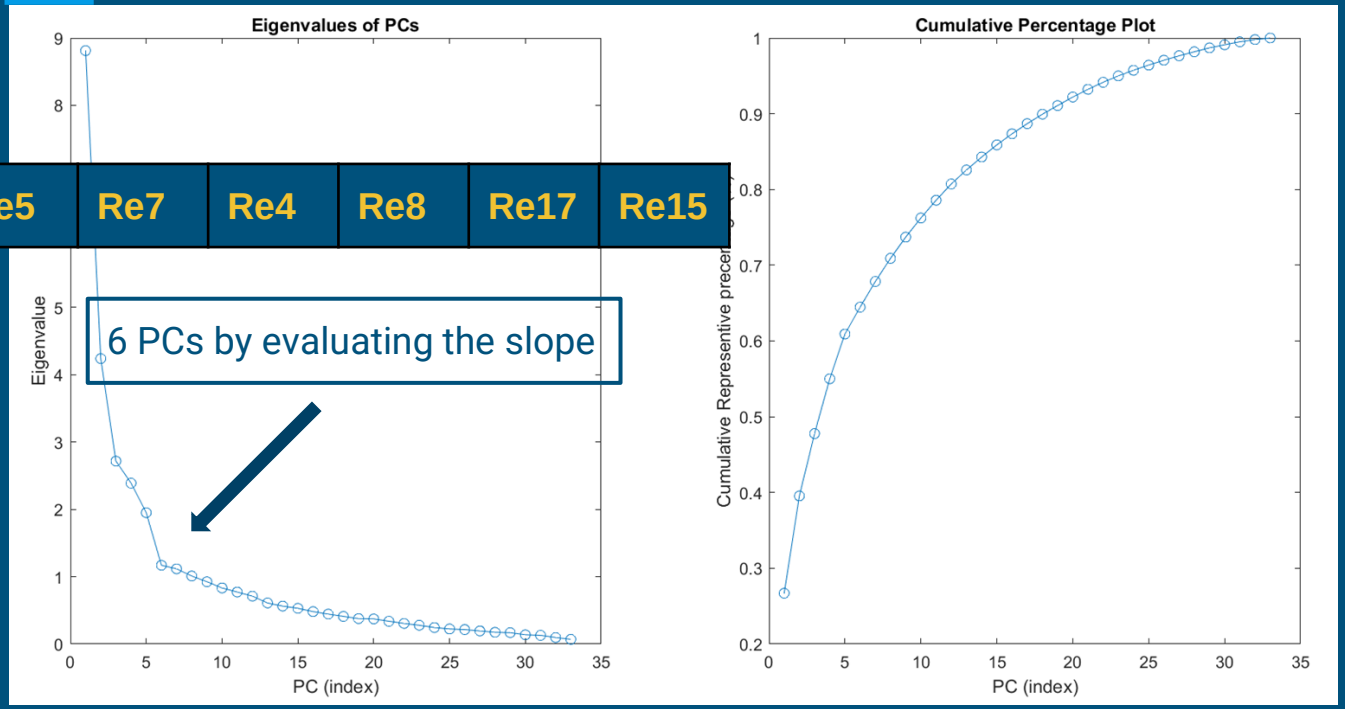
[1] V. G. Sigillito, S. P. Wing, L. V. Hutton, K. B. Baker, 1989, *Classification of radar returns from the ionosphere using neural networks.*

PCA: graphical analysis

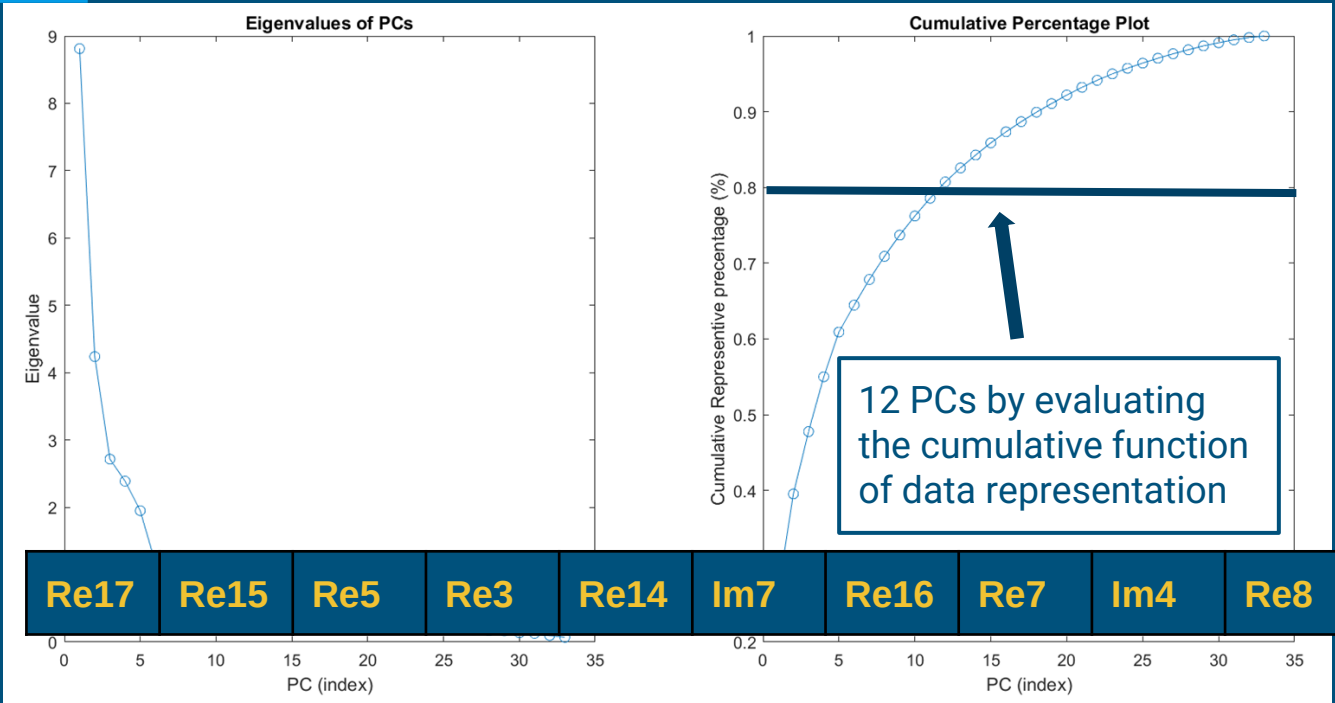




PCA: graphical analysis

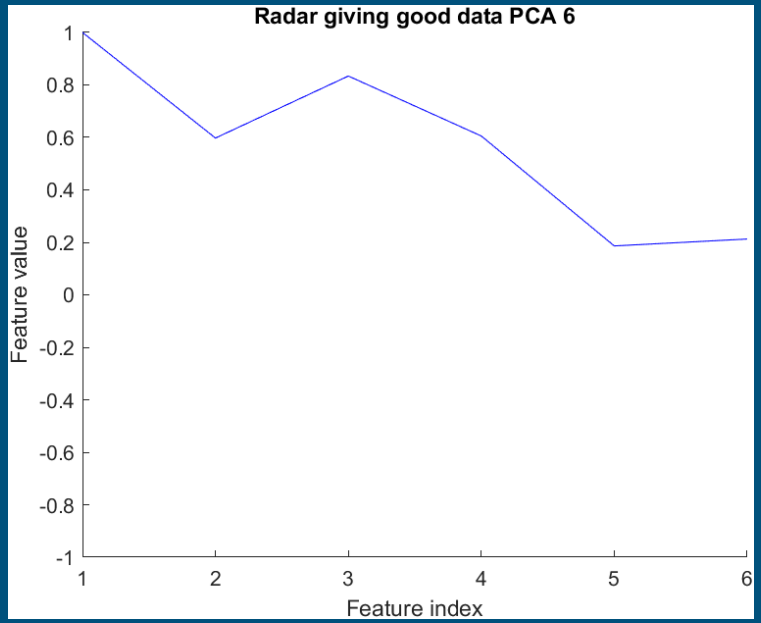


PCA: graphical analysis

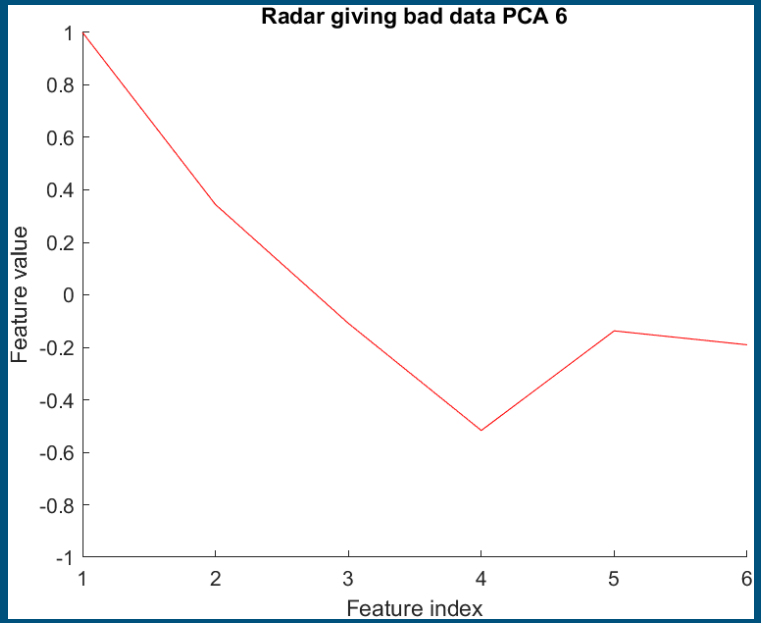


Re17	Re15	Re5	Re3	Re14	Im7	Re16	Re7	Im4	Re8	Re4	Re13

6 PCs : return to the features space

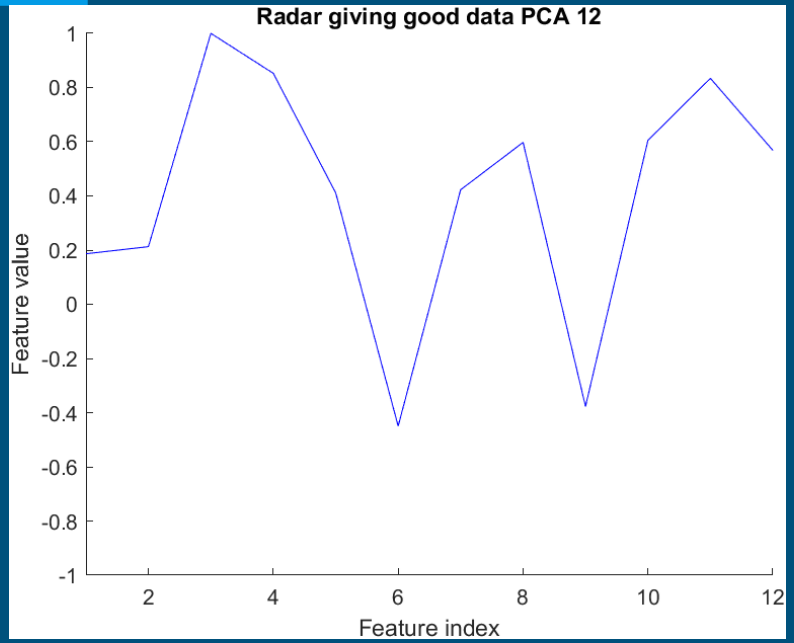


Examples of good data returns

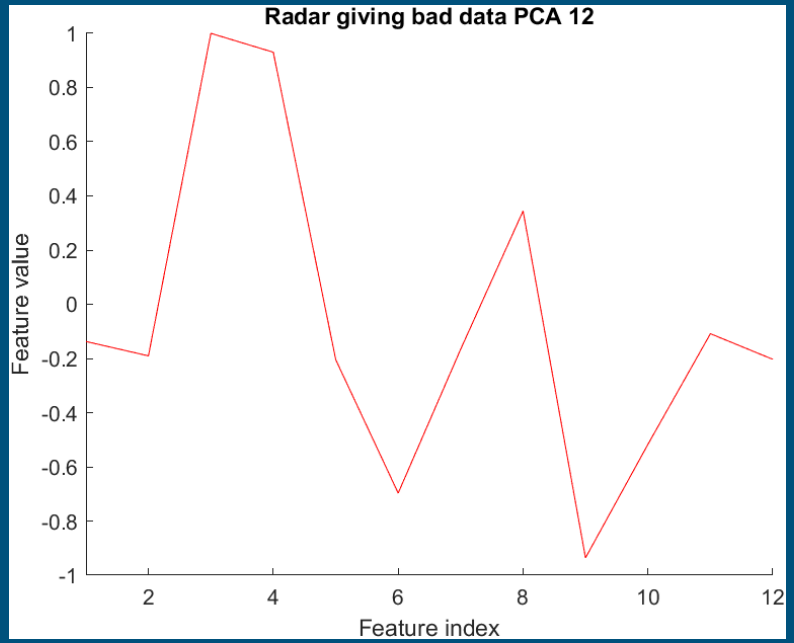


Examples of bad data returns

12 PCs : return to the features space



Examples of good data returns



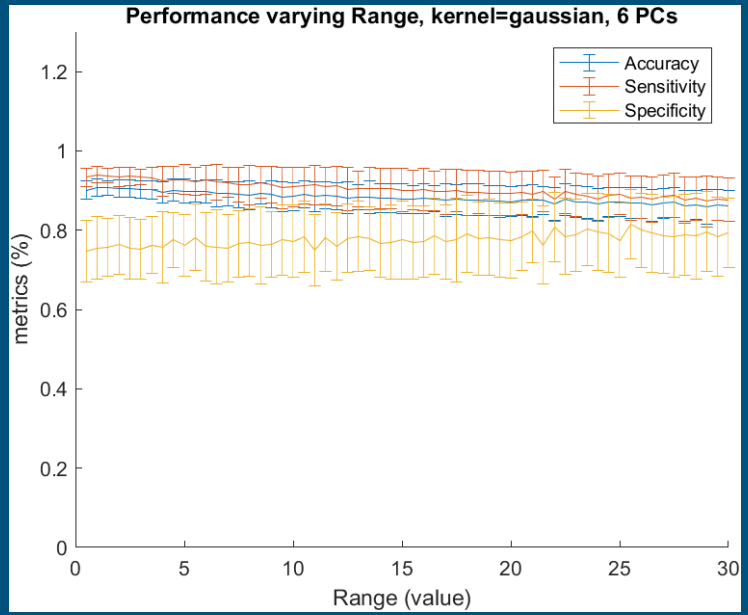
Examples of bad data returns

PCA: implementation in SVM algorithm

Some runs were performed with the same set of hyperparameters in the SVM code but we considered just the most representative features.

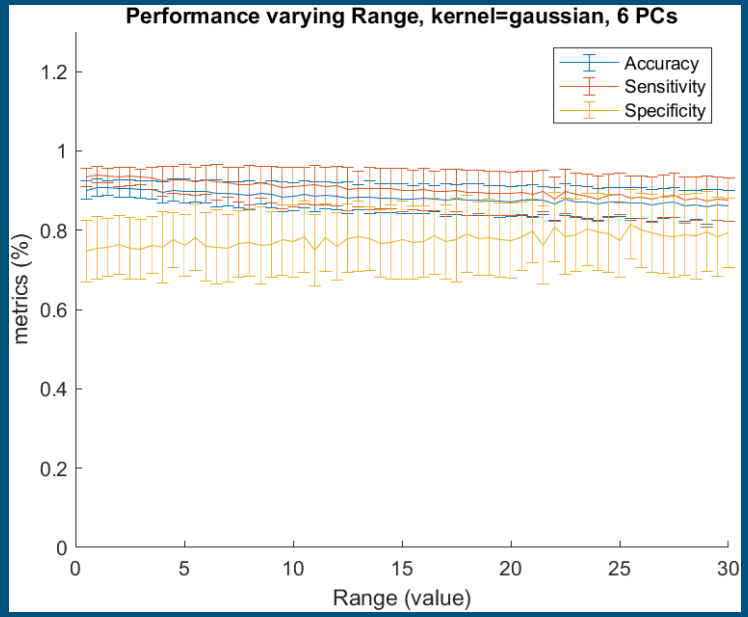
PCA: implementation in SVM algorithm

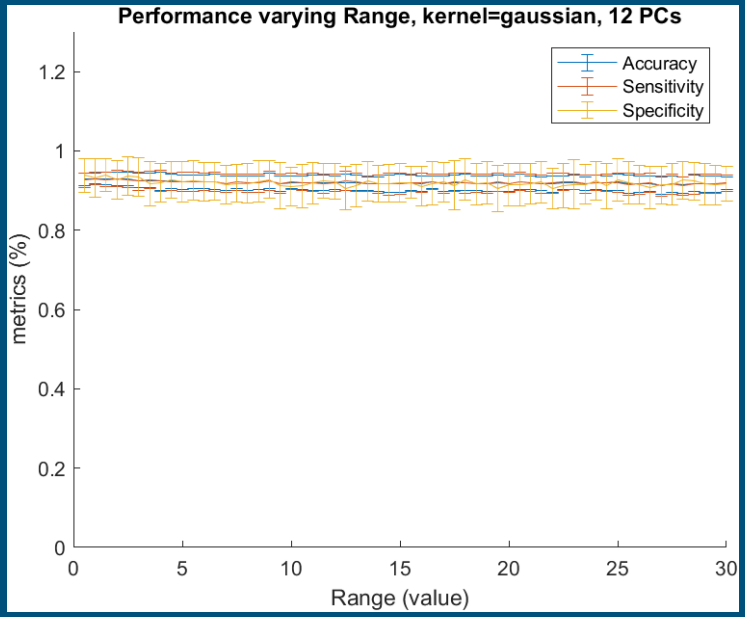
Some runs were performed with the same set of hyperparameters in the SVM code but we considered just the most representative features. With gaussian kernel we found an interesting behaviour.



PCA: implementation in SVM algorithm

Some runs were performed with the same set of hyperparameters in the SVM code but we considered just the most representative features. With gaussian kernel we found an interesting behaviour.

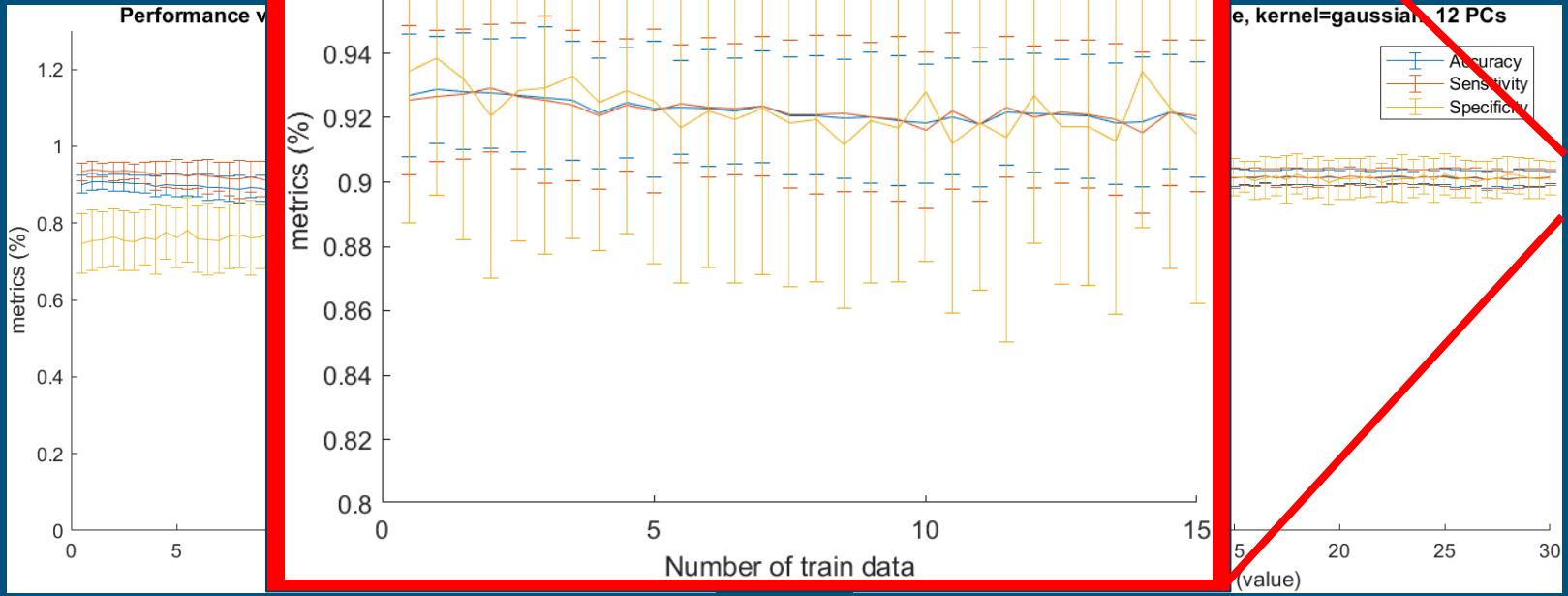




PCA: implementation in SVM algorithm

Some runs were p
just the most repr

code but we considered
sting behaviour.



SVM with PCA: best performances

TPR mean	TPR std	TNR mean	TNR std	FPR mean	FPR std	FNR mean	FNR std	acc mean	acc std	Range
0,929920	0,015964	0,931923	0,047925	0,068077	0,047925	0,070080	0,015964	0,930265	0,013193	1,0
0,928080	0,018976	0,938846	0,040577	0,061154	0,040577	0,071920	0,018976	0,929934	0,015728	1,5
0,928160	0,019922	0,936154	0,048963	0,063846	0,048963	0,071840	0,019922	0,929536	0,017344	2,5
0,930000	0,020916	0,926923	0,049654	0,073077	0,049654	0,070000	0,020916	0,929470	0,016822	2,0
0,926640	0,018372	0,938462	0,042345	0,061538	0,042345	0,073360	0,018372	0,928675	0,015115	0,5

SVM with PCA: best performances

SVM gaussian with PCA 12 - time ≈ 4.10 sec

TOTAL SAMPLES 151	ACTUAL POSITIVE 125	ACTUAL NEGATIVE 26
PREDICTED POSITIVE 118 ± 3	TRUE POSITIVE 117 ± 2	FALSE POSITIVE 2 ± 1
PREDICTED NEGATIVE 32 ± 3	FALSE NEGATIVE 8 ± 2	TRUE NEGATIVE 24 ± 1
ACCURACY 93.31% ± 1.60%	SENSITIVITY 93.32% ± 1.91%	SPECIFICITY 93.27% ± 4.99%

SVM polynomial RANGE=29 - time ≈ 4.5 sec

TOTAL SAMPLES 151	ACTUAL POSITIVE 125	ACTUAL NEGATIVE 26
PREDICTED POSITIVE 122 ± 7	TRUE POSITIVE 116 ± 5	FALSE POSITIVE 6 ± 2
PREDICTED NEGATIVE 29 ± 7	FALSE NEGATIVE 9 ± 5	TRUE NEGATIVE 20 ± 2
ACCURACY 89.83% ± 3.05%	SENSITIVITY 92.67% ± 3.63%	SPECIFICITY 76.15% ± 7.75%

Multiclassifier with our 3 “best algorithms”

We perform a run with a multiclassifier created by taking the best-performing setup for each algorithm so we have:

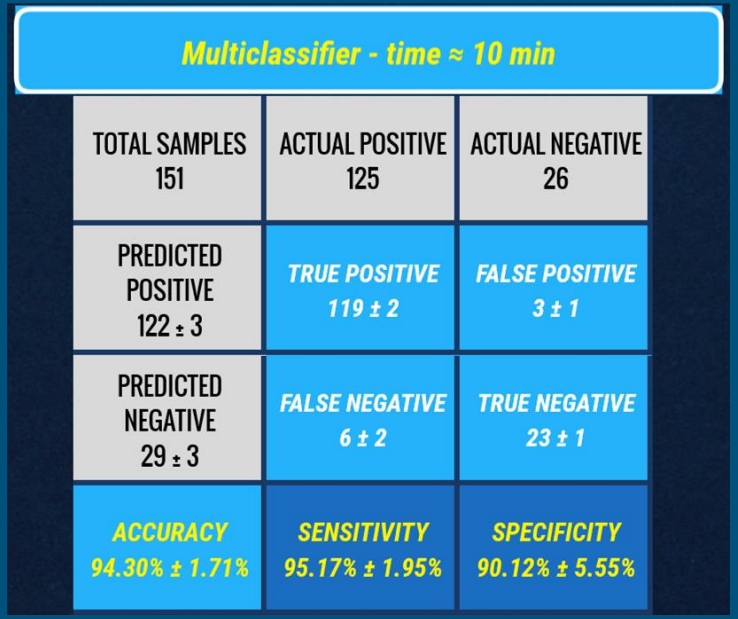
- SVM with 12 PCs, gaussian kernel and range=1; → Acc = 93.31%, Sen = 93.32%, Spe = 93.27%
- Random Forest with 50 trees; → Acc = 92.85%, Sen = 93.50%, Spe =89.97%
- Simple Neural Network with LR=0.93 and 1000 epoch.[1] → Acc = 90.81%, Sen = 98.35%, Spe =56.22%

(1) To be completely transparent we used a slightly different setup in this case, we actually use the custom “fold” for data selection

Multiclassifier with our 3 “best algorithms”

We perform a run with a multiclassifier created by taking the best-performing setup for each algorithm so we have:

- SVM with 12 PCs, gaussian kernel and range=1;
- Random Forest with 50 trees;
- Simple Neural Network with LR=0.93 and 1000 epoch.(1)



(1) To be completely transparent we used a slightly different setup in this case, we actually use the custom “fold” for data selection

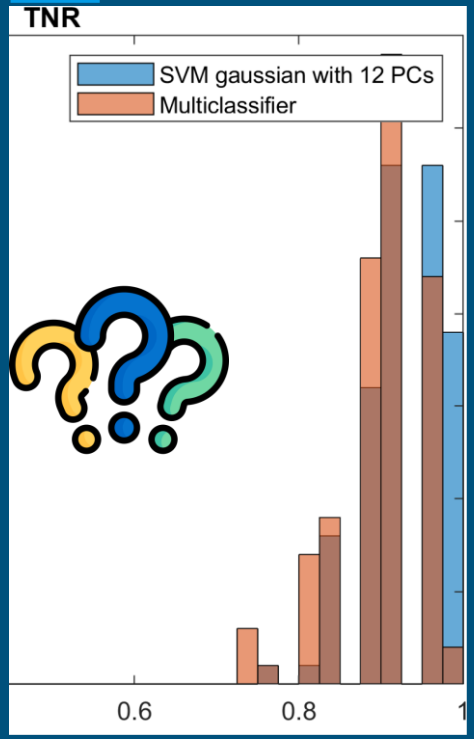
The best algorithm: Kolmogorov-Smirnov test

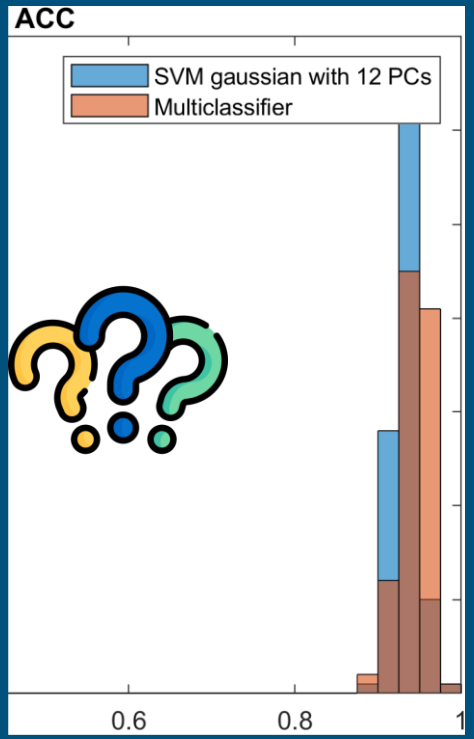
By using the KS test(2) we compare our best performances algorithm in order to see if we can find statistically relevant differences.
From matlab documentation we can read:

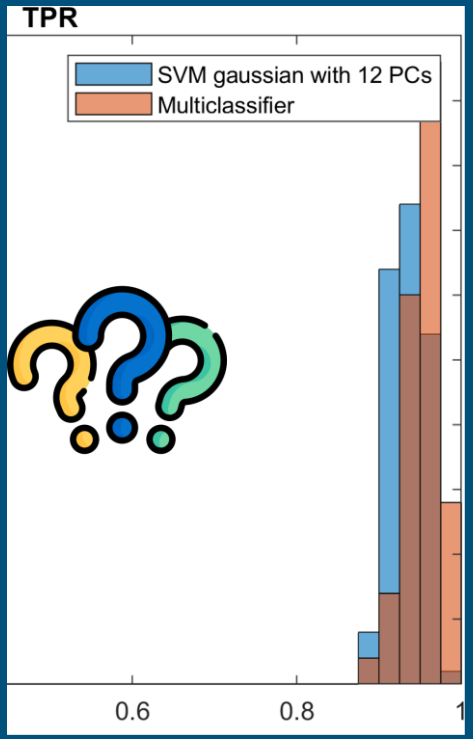
h = kstest2(x1,x2) returns a test decision for the null hypothesis that the data in vectors x1 and x2 are from the same continuous distribution, using the two-sample Kolmogorov-Smirnov test. The alternative hypothesis is that x1 and x2 are from different continuous distributions. The result h is 1 if the test rejects the null hypothesis at the 5% significance level, and 0 otherwise

(2) The test was performed also for other comparisons but we can appreciate its importance in this particular case

The best algorithm: Kolmogorov-Smirnov test







The best algorithm: Kolmogorov-Smirnov test

h = kstest2(x1,x2) returns a test decision for the null hypothesis that the data in vectors x1 and x2 are from the same continuous distribution, using the two-sample Kolmogorov-Smirnov test. The alternative hypothesis is that x1 and x2 are from different continuous distributions. The result h is 1 if the test rejects the null hypothesis at the 5% significance level, and 0 otherwise

```
Set    Null_val    pvalue
TPR      1        0.000003
TNR      1        0.008216
FPR      1        0.008216
FNR      1        0.000003
acc      1        0.000092

 --- Guardo media e std delle metriche dei due ---
Set    Mean pm std
TPR1   0.933200 pm 0.019141    TPR2        0.951680 pm 0.019527
TNR1   0.932692 pm 0.049916    TNR2        0.901154 pm 0.055549
FPR1   0.067308 pm 0.049916    FPR2        0.098846 pm 0.055549
FNR1   0.066800 pm 0.019141    FNR2        0.048320 pm 0.019527
acc1   0.933113 pm 0.016043    acc2        0.942980 pm 0.017122
```

The best algorithm: Kolmogorov-Smirnov test

h = kstest2(x1,x2) returns a test decision for the null hypothesis that the data in vectors x1 and x2 are from the same continuous distribution, using the two-sample Kolmogorov-Smirnov test. The alternative hypothesis is that x1 and x2 are from different continuous distributions. The result h is 1 if the test rejects the null hypothesis at the 5% significance level, and 0 otherwise

```
Set     Null_val     pvalue
TPR        1        0.000003
TNR        1        0.008216
FPR        1        0.008216
FNR        1        0.000003
acc        1        0.000092

 --- Guardo media e std delle metriche dei due ---
Set     Mean pm std                 
TPR1    0.933200 pm 0.019141    TPR2        0.951680 pm 0.019527
TNR1    0.932692 pm 0.049916    TNR2        0.901154 pm 0.055549
FPR1    0.067308 pm 0.049916    FPR2        0.098846 pm 0.055549
FNR1    0.066800 pm 0.019141    FNR2        0.048320 pm 0.019527
acc1    0.933113 pm 0.016043    acc2        0.942980 pm 0.017122
```

So… What is the best choice ?

SVM gaussian with PCA 12 - time ≈ 4.10 sec

TOTAL SAMPLES 151	ACTUAL POSITIVE 125	ACTUAL NEGATIVE 26
PREDICTED POSITIVE 118 ± 3	*TRUE POSITIVE 117 ± 2*	*FALSE POSITIVE 2 ± 1*
PREDICTED NEGATIVE 32 ± 3	*FALSE NEGATIVE 8 ± 2*	*TRUE NEGATIVE 24 ± 1*
ACCURACY 93.31% ± 1.60%	*SENSITIVITY 93.32% ± 1.91%*	*SPECIFICITY 93.27% ± 4.99%*

Multiclassifier - time ≈ 10 min

TOTAL SAMPLES 151	ACTUAL POSITIVE 125	ACTUAL NEGATIVE 26
PREDICTED POSITIVE 122 ± 3	*TRUE POSITIVE 119 ± 2*	*FALSE POSITIVE 3 ± 1*
PREDICTED NEGATIVE 29 ± 3	*FALSE NEGATIVE 6 ± 2*	*TRUE NEGATIVE 23 ± 1*
ACCURACY 94.30% ± 1.71%	*SENSITIVITY 95.17% ± 1.95%*	*SPECIFICITY 90.12% ± 5.55%*

Ideas for future improvements

- Creating a multi-layer neural network and test it with a balanced train data set as the one seen before ;
- Exploring better the PCA technique in all algorithms(3) ;
- Improving explainability by graphical simplified representation of our trained models ;
- Creating a PCA method that works in the PCs' space instead of the features' space;
- Creating a complete K-Fold method and try it on our algorithms ;
- Exploring better the limits of our algorithm by reducing and unbalancing train(4) ;

(3) We have tried a single run with 12 PCs with all algorithms but didn't analyzed them properly

(4) We have tried to reduce training data in a range from 10 to 200 for our "bests algorithm"