Datasets choice

1- First Dataset:

The first dataset is Indian patient records in and their binary labels as having or not a liver disease.
The features used are basic patient information (age, gender), levels of some chemical compounds present in the human body and results of some medical tests.
The datasets size is : 583 data points with 10 features.
The labels ratios are : 72% positive examples versus 28% positive examples.
Meaning that it will be also very important to evaluate recall and precision (we'll use F1 score).

2- Second Dataset:

The second dataset is for fall detection, it is a multi-class dataset that labels elderly people in a chinese hospital, given some of their monitored health data (Sugar level, heart rate, blood pressure etc…) in six different states : Standing, Walking, Sitting, Falling, Cramps or Running.
The datasets size is : 16382 datapoint with 6 features.
The ratios for these states are respectively : 28% 3% 15% 22% 22% 10%.
Which is also not completely balanced, we'll remember that "Walking" state has a very small ratio.

3- Motivation:

What makes these two datasets interesting to me is first of all, as falls are one of the major health issues that elderly face, especially when they're not permanently monitored by nurses. And also since liver damage has been an increasing health issue in India due to the excessive consumption of alcohols and/or contaminated foods etc..

This makes our task that we have at hand meaningful and potentially very impactful. But what's also important is that these datasets are perfect for evaluating different machine learning algorithms for the following reasons:

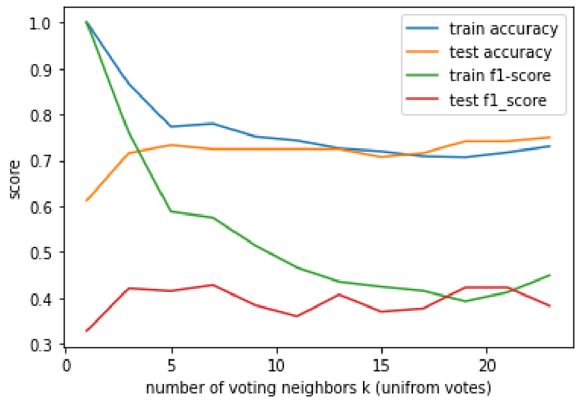
- A lot of relevant features for both datasets (We can be fairly confident that the information of the liver's health is somewhat included in the medical tests and chemical levels, and similarly the state of the elederly person given their heart rate etc..
- Clean datasets and almost no preprocessing needed
- The datasets are of varying lengths, so we'll see what's optimal in terms of splitting validation etc.. depending on dataset size.
- Binary classification vs multi-class classification with the same algorithms.

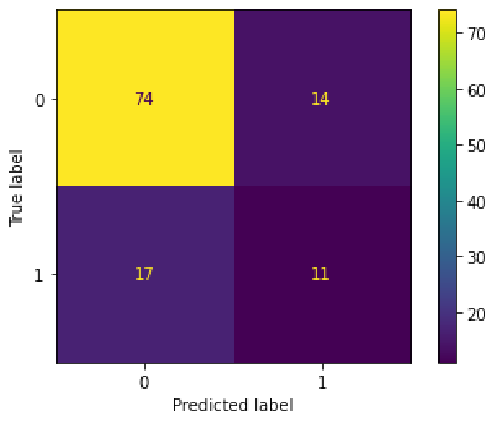
Algorithms Implementation and evaluation

1.KNN

a.Dataset 1

The first step is the tuning of the model, we'll try to settle for a number of neighbors k and

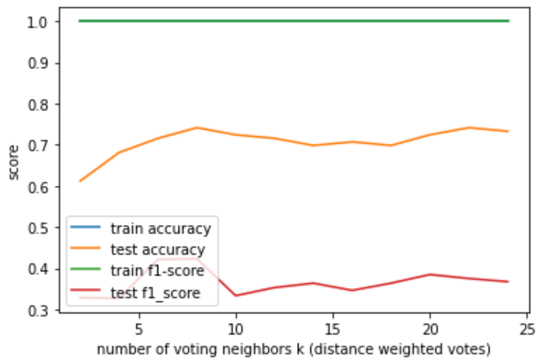




we'll also try weighting the votes with two kernels. Then we'll plot the learning curves to see how the data size impacts the training of KNNs. and we'll also plot the confusion matrix.

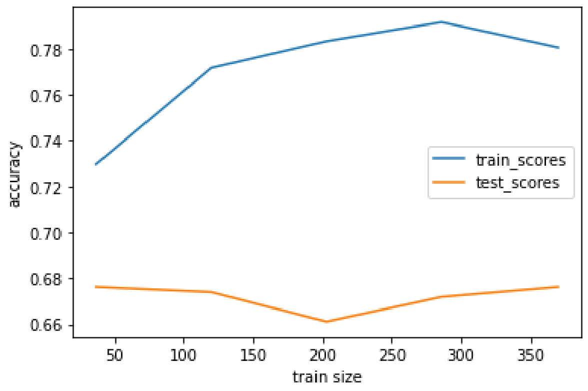
As "k" increases, the bias increases and variance decreases, which we can notice by comparing train and test scores. (bias 0 for k=1 since every point votes for itself)

We can also see that when we weigh our votes variance decreases without really increasing the bias (but we do not get better test score)



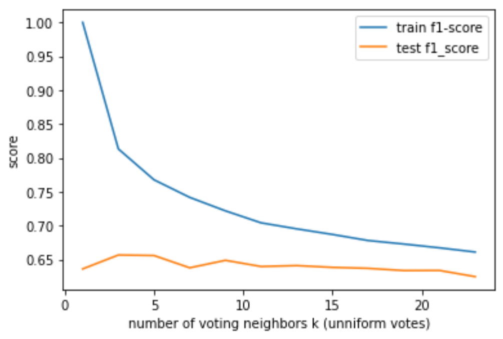
We'll settle for k= 5. (even if k=25 has a better accuracy because k=25 for a train set of size 400 induces very high variances, so the results are not reliable). **Training f1 score : 1 Test F1 score :0.4**

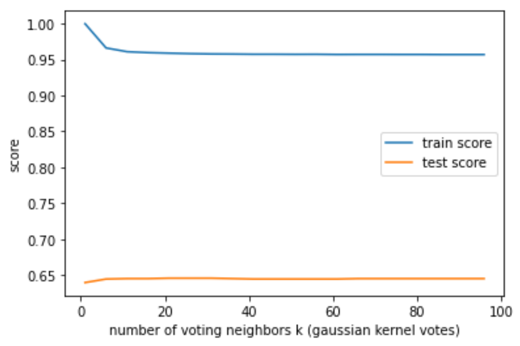
The recall is still insufficient for label 1 due to class ratio disparity, but it still does a better job than a naive classifier (always predict predominant class)



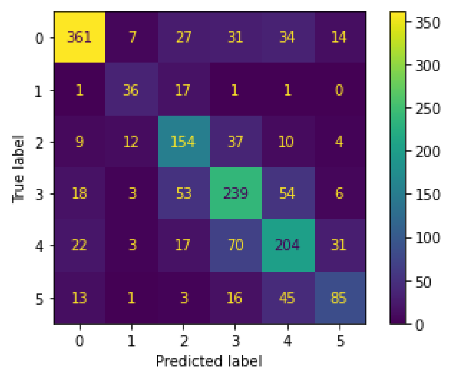
Our learning curve seems to be increasing with sample numbers, so we can assume that with k=5 our model could use more data. Luckily our second dataset has a lot more data.

b.Dataset 2





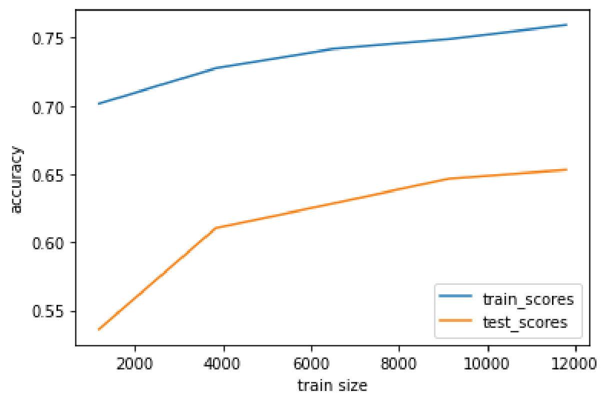
We can notice the same things as for the first dataset, variance decreases with k. And k=5 seems once again to be a good choice. **Training precision : 0.95 Test precision : 0.65** We also tried another gaussian kernel to smooth out the votes, but all it did once was reduce bias without really improving our test accuracy that remains around 65%



Now looking at the confusion matrix, we see that class disparity is less of a problem although for the class labeled as “1” only represents 3% of the data.

The diagonals are always the highest both in their rows and columns, meaning both the precision and recall for every class are decent. Two things could explain this:

- Non trivial data size : 3% of 16000 data point is still a relevant amount of data in our problem’s scale
- Relatively low number of features (only 6 for 16 000 data points) meaning the information is somewhat easier to retrieve than when we have 500 points for 11 features like in dataset 1 (This is a very vague speculation as it isn’t perfectly pertinent to compare two different datasets)

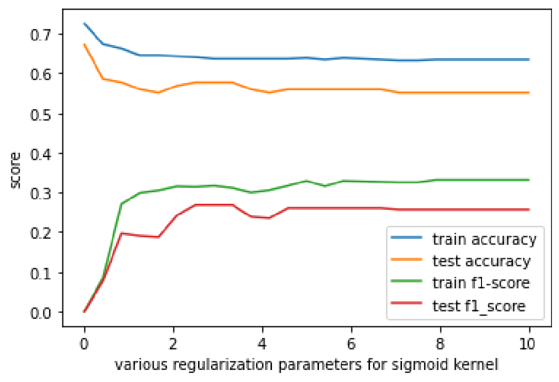


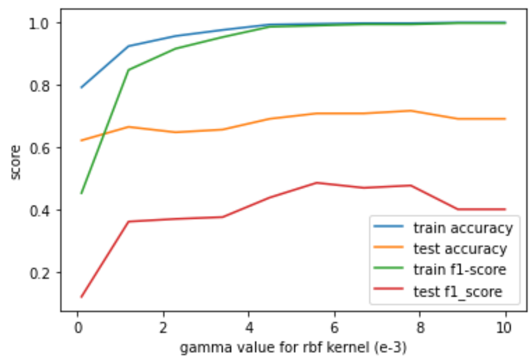
The learning curve is still increasing with sample numbers at the end of our dataset but we specifically chose our k=5 in this case cause we saw that the model wouldn’t overfit.

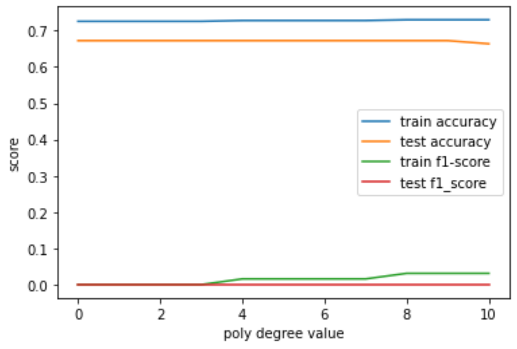
Notice that in the first curve, even for higher k values, the test accuracy remains stable meaning that the size of our dataset moderates the model’s variance.

2.SVM

a.Dataset 1



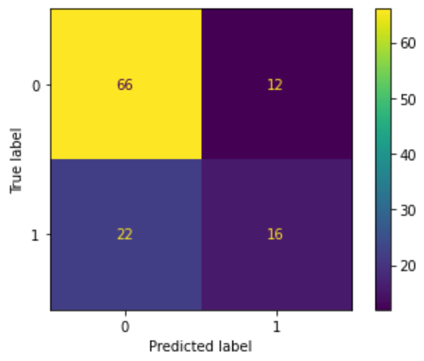




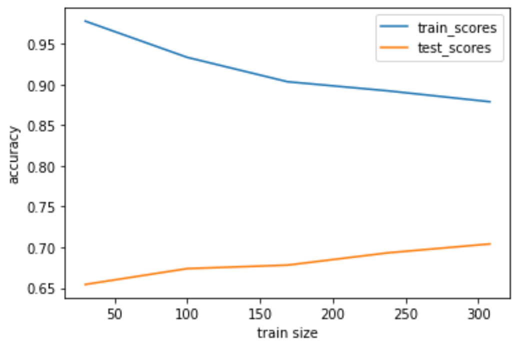
For the SVM we tuned the regularization (penalty parameter) with a sigmoid kernel, then we tried a polynomial kernel (tuned with degree) and rbf kernel (that we tuned with gamma)

Gamma controls how much influence a single example can have and C how much we penalize a misclassification (hence the need for a low gamma and high C here).

. This required a lot of tuning, because it is very easy for the model to just decide to classify everything as the predominant class (High precision low recall).
Based on those tests we'll continue with rbf kernel of value gamma = 0.0006.
Training f1 score : 0.9 Test F1 score :0.45

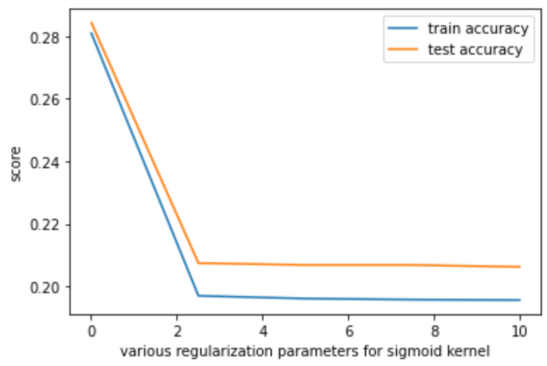


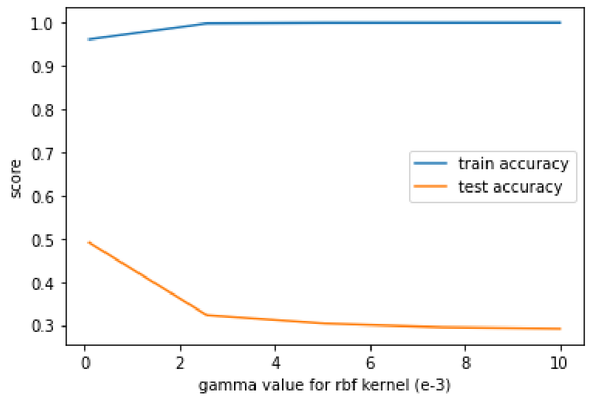
The confusion matrix (figure on the left) for this tuning shows the recall for the low ratio class.



The learning curve indicates that there is no overfitting or underfitting, but it also shows that the model does learn a little bit since with increasing samples, the training and testing scores converge to a line in the middle.

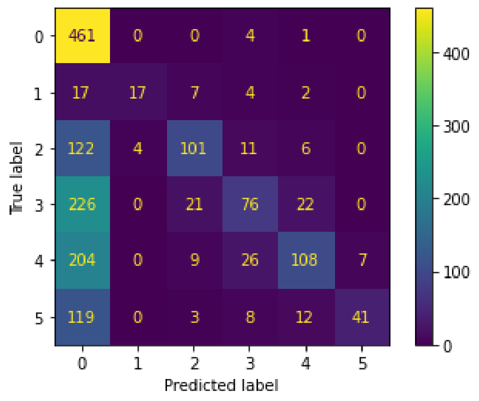
b.Dataset 2



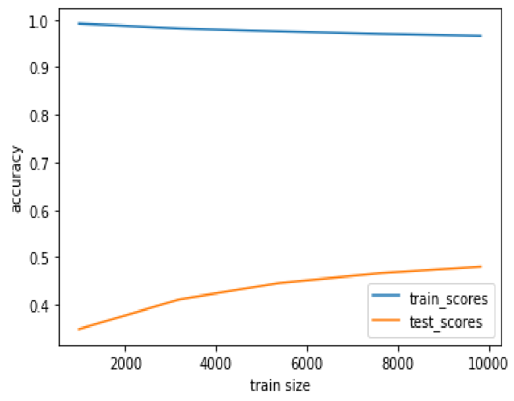


The multi-class for the chosen library (sklearn) is implemented using One-to-One binary SVM. Meaning the complexity increases exponentially with class numbers. So the runtimes for these training were a lot slower.

We can find the same issue as for dataset 1, it is hard to linearly separate the data. Especially since the data set is not very balanced. We need a very strong penalty to dissuade the model from just classifying everything as the predominant class.
So as we see for the rbf kernel we need to take a very small value for gamma.
And then we can tune a rbf kernel (gaussian) to find the best score at gamma = 0.0001
Training f1 score : 1 Test F1 score :0.5



The confusion matrix explicits how the model struggles with recall. Even when we get gamma lower and C higher, our SVM still somewhat follows the global trend of the data
So the tradeoff is between regularization and smoothness of the separation plane.



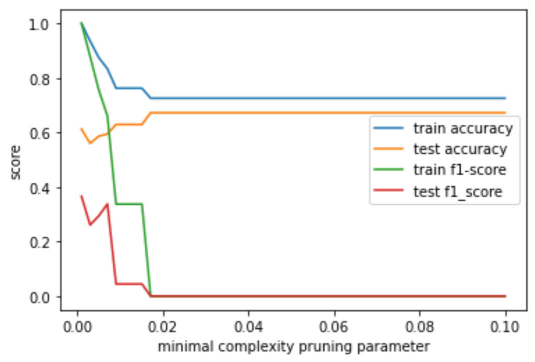
We can see a strong offset between training and testing score, it seems to overfit while in reality we are “forced” to overfit a little bit, because if we don’t, the model will tend to misclassify the classes with few examples a lot more.

This is the hardest model to tune, since it is extremely sensitive to many hyperparameters (at least C, and gamma if we don’t talk about the kernels).
All while having very high computational complexity.

3.Decision Tree

a.Dataset 1

This implementation of decision trees is using the GINI criterion to calculate nodes impurity

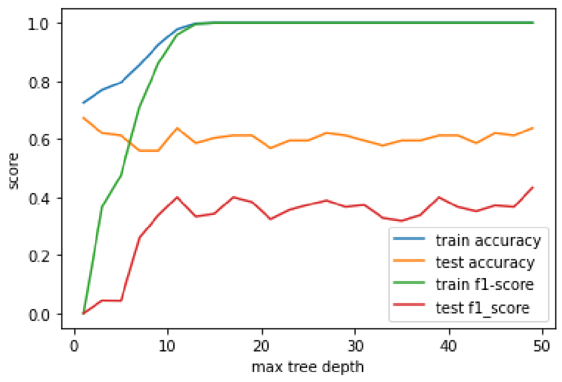


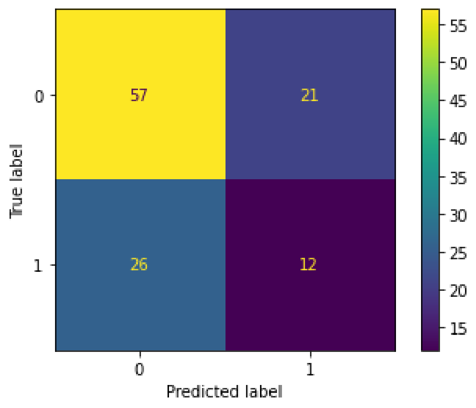
We first tried tuning this ccp parameter, it implements minimal-cost complexity pruning. Meaning trees with a lower complexity than this constant will be pruned. Therefore the model will generalize more and overfit less.

We also tuned the max depth parameter, which forces the model to stop splitting attributes when it gets to a certain tree, default is infinity which is why the model trains to have 0 bias, but fixing a max length can also help avoid overfitting0

Based on the experimental results we'll settle for a max length of 15.

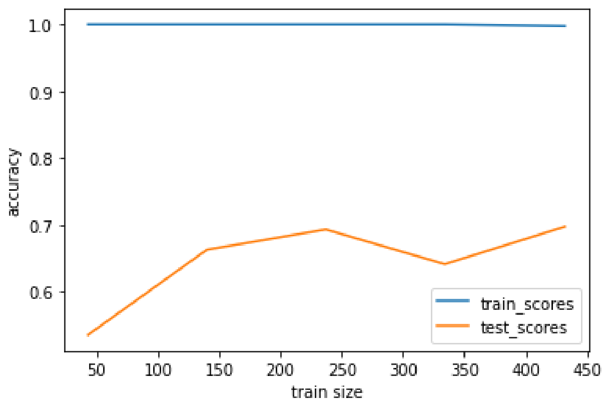
Training f1 score : 1 Test F1 score :0.4



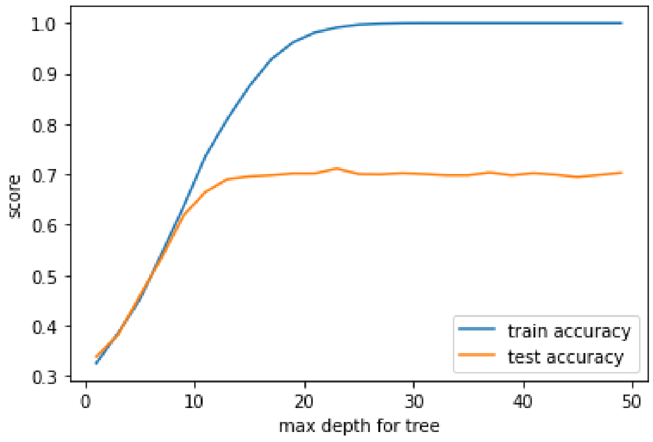


We can see the low f1 score on the confusion matrix as the second class is very often mislabeled. This is a recurrent problem for this dataset. It is hard to avoid overfitting without forcing the model to neglect the class with lower ratio.

The learning curve on the right shows our testing accuracy does increase with our data size and that it has not converged yet at the end of our dataset.
Which also indicates that we could achieve greater results with more data.



b.Dataset 2



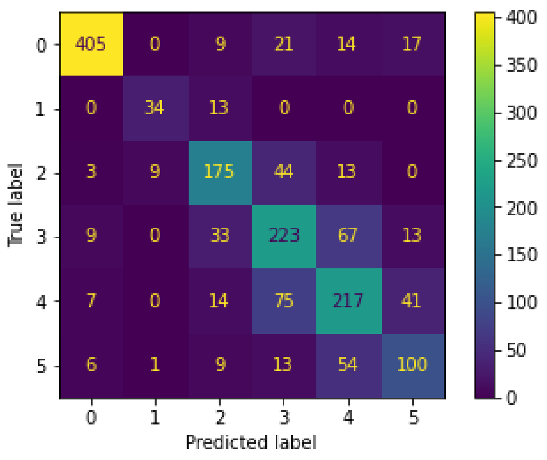
For this dataset, we'll use max depth to stop our trees from overfitting, the curve on the left shows the evolution of train and test accuracy depending on the max depth.

It doesn't seem that model overfits anyway even for an infinite max depth.

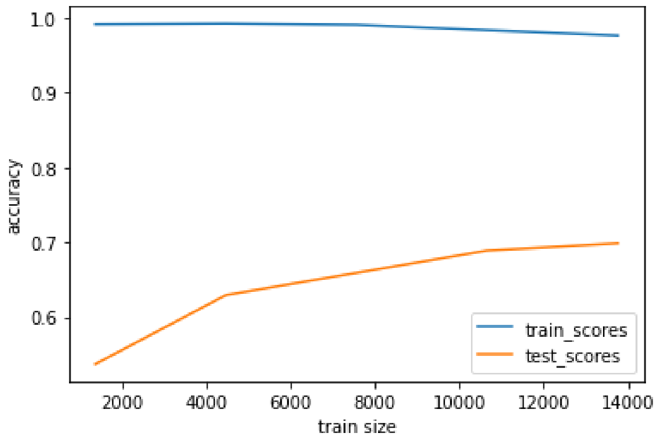
Training f1 score : 1 Test F1 score :0.7

This is one of the best matrices we got for this dataset yet. The diagonal values are the strongest (even for the label 1 which represents only 3% of the dataset).

That means that this classifier does well both on precision and recall.



	0	1	2	3	4	5
0	405	0	9	21	14	17
1	0	34	13	0	0	0
2	3	9	175	44	13	0
3	9	0	33	223	67	13
4	7	0	14	75	217	41
5	6	1	9	13	54	100



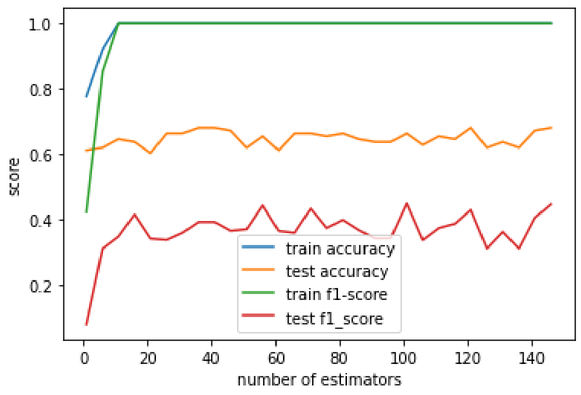
This learning curve is very similar to the one for dataset1 with the difference that it seems this one started to converge. Which makes sense since we have a much larger dataset than for dataset1.

We reach an overall precision of 70%.

4.Boosted (AdaBoost) decision trees

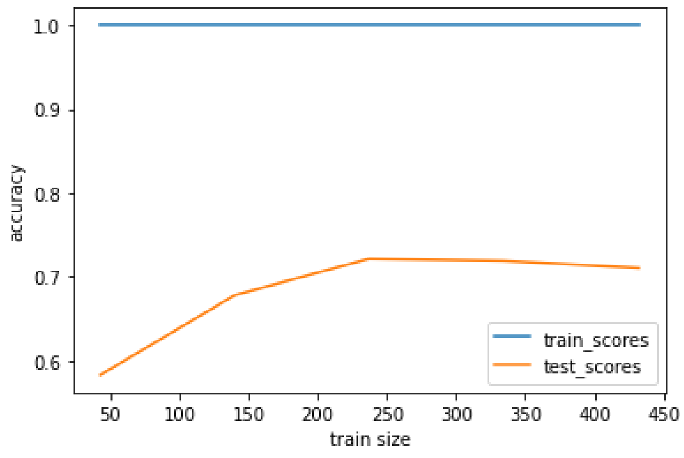
a.Dataset 1

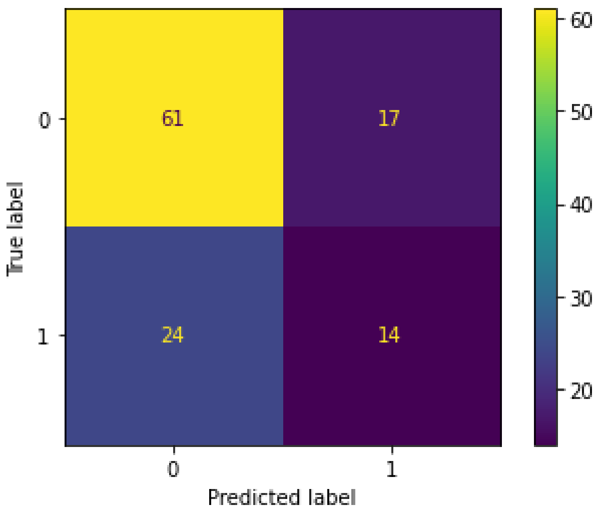
AdaBoost is a meta estimator, that boosts a weak estimator with the particularity of focusing on at every step on the data points where the error was highest on the previous step.



We're using decision trees as weak learners (we set the max depth to be relatively low, to make sure the weak learners have low variance)
The figure on the left shows the tuning of our AdaBoost on the number of estimators used.

Although very noisy, we'll settle for k = 25
Training f1 score : 1 Test F1 score :0.4

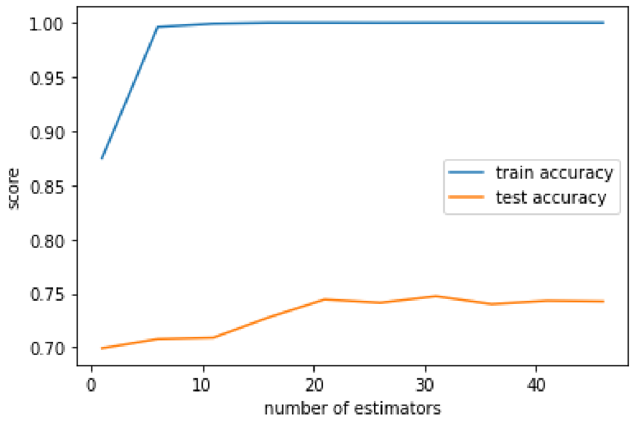




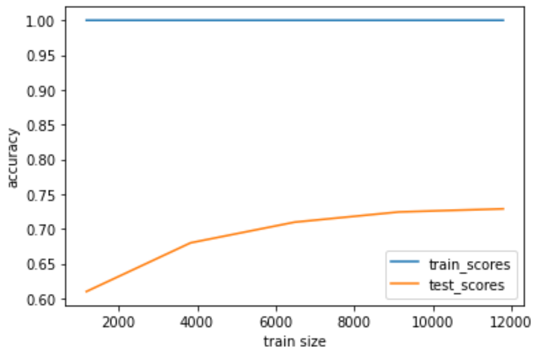
Learning curve and confusion matrix are very
similar to when we used only one decision tree, and even tuning the learning rate for the boosting does not help. This simply means that the boosting is not effective for our dataset which makes sense, since boosting usually works better when we struggle with decreasing the bias . For this particular dataset, the problem is the variance (most of the other classifiers that we tried so far were able to achieve 0 bias, but all of their learning curves seems to be still far from converging.

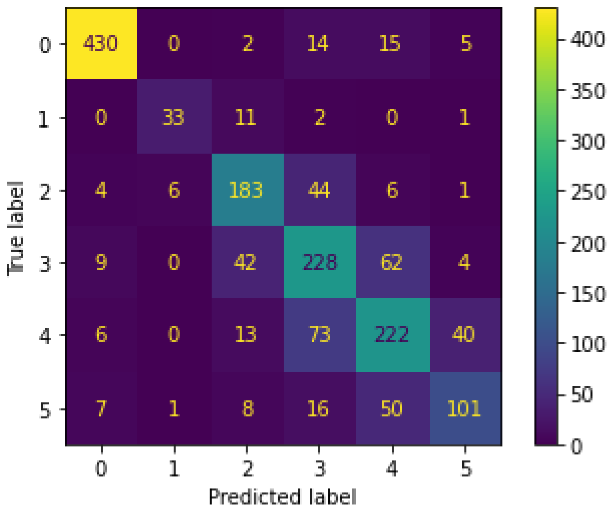
b.Dataset 2

Applying the AdaBoost to this dataset gives us the best precision among the other algorithms.



The figure on the left shows the tuning for the estimator numbers.
For k = 20 we reach:
Training precision : 1 Test precision :0.75





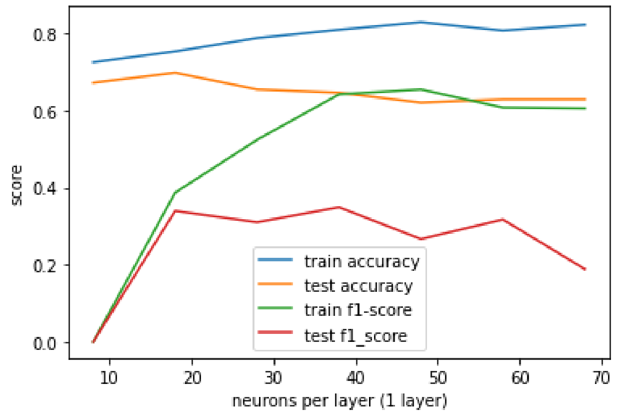
We can see thanks to the confusion matrix that the model performs well for every single label.
We already had a decent bias with the decision trees, so it is a bit surprising that boosting was effective on increasing the score. But this could be due to the fact that boosting allows us to be more aggressive with the tuning of the hyperparameters.

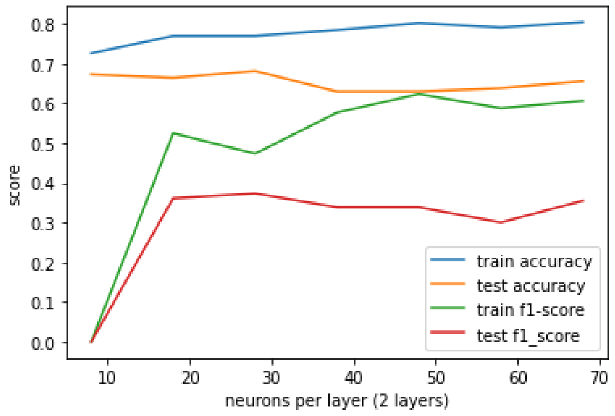
The learning curve appears to have reached its limit, so we can speculate that the accuracy wouldn’t get much higher than this even if changed the tunings or increased our data size.

5.Neural Network (MLP)

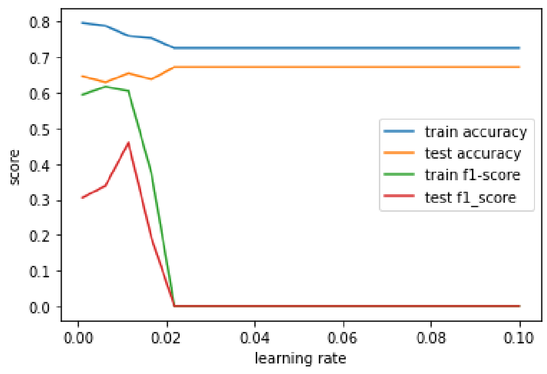
a.Dataset 1

This classifier is a ANN, implemented with a logarithmic loss and a stochastic gradient descent with momentum (adam). We also tried this classifier on augmented (just repetitions) Versions of our dataset. (equivalent to epochs)



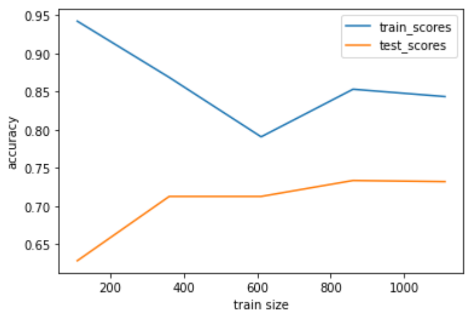


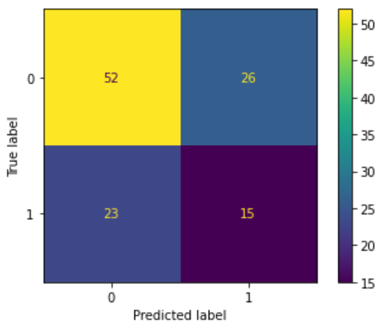
The first thing to tune is the number of neurons and of layers. We tried 1 then 2 layers with an increasing number of neurons (two figures above). Neural networks generally induce higher risk of overfitting due to their high number of learnable parameters. So this is probably not the best choice for a dataset that struggles with high variance with simple classifiers.



We also try to tune on the learning rate.
We find a little optimum at lr= 0.015.

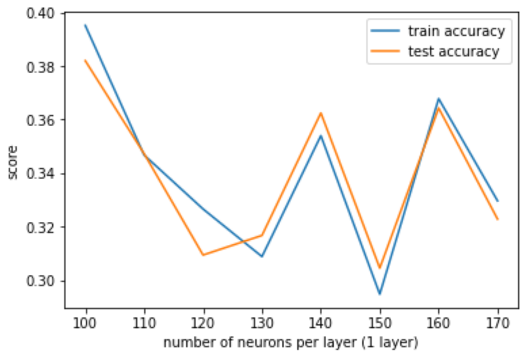
We keep the model with 3 Layers of 50 neurons each
Training f1 score : 0.6 Test F1 score :0.4

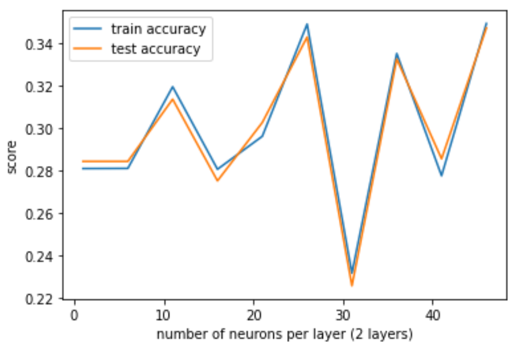




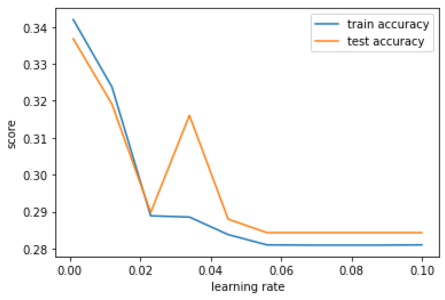
The confusion matrix shows that once again the model just learned to overpick the predominant class.
In cases like this, the learning curve evaluating the precision is not very relevant.

b.Dataset 2



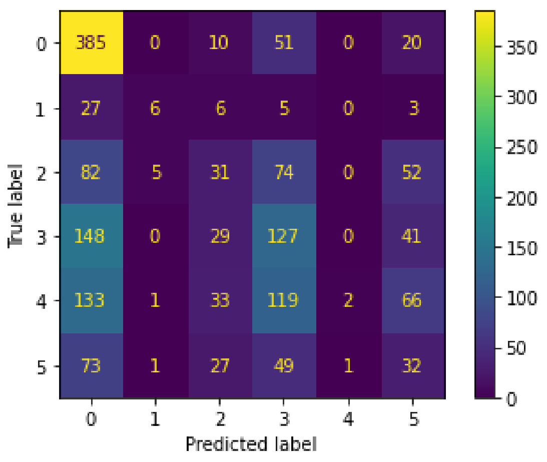


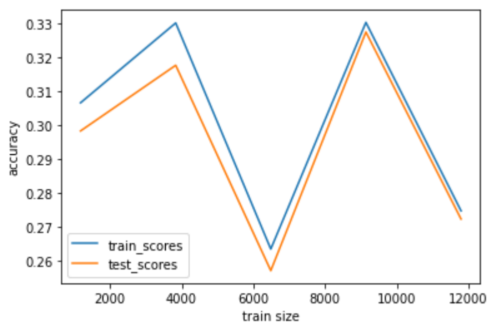
We go through the same tuning steps as for dataset 1, we can see the evolution of the precision scores with number of neurons for 1 or 2 layers (figures above)
And with the learning curve (Figure below)



This is the worst model for this dataset, but we'll settle for 2 Layers with 32 neurons each and a learning rate of 0.001 for **Training precision : 0.35 Test precision :0.35**

We can see thanks to the confusion matrix that a lot of misclassification is happening.
It might be that the problem is too simple for a neural network, we saw that simple classifiers could give decent precision values.
There is a certain degree of linearity between the features and the target vectors. Which is overlooked by using a neural net.





The learning curve is noisy, we can see these small fluctuations (between 0.26 and .33 accuracy) going on.

Algorithms comparison: Conclusion

Trying all these algorithms on both these datasets allowed us to get a lot of insight on the data itself.

Although dataset 2 is a multi-class problem, it is much simpler to achieve good scores on it. It has a decent size, and we can safely conclude that, for the most part, the information of the target class is relatively easy to find from the predictors.

We got the best result on this dataset with the adaboost algorithm, which constitutes a good choice. SVM and neural networks on the other hand are not ideal:

- SVMs computational complexity increases a lot with class numbers (and especially if we want to try higher degree polynomial kernels) and
- ANNs are best suited for complex datasets where patterns are very hard to find which is not the case for this dataset (for instance just the heartrate alone gives a lot of information on whether the person is standing, or falling etc..)

Dataset 1 being a binary classification suffered from a small size which caused all the models to have a very high variance on it. On top of that, the disparity between the two classes and the high number of features did not help.
But SVMs still make a good choice for this model since we can decide on a penalty to help us find a smooth separation between the two classes, without having too much extra computational cost.
Three things that could help achieve better results on this dataset are :

- Increase the data size (although not always possible).
- Use feature selection to get rid of information redundancy and have a lower more relevant features number.
- Try to balance the dataset by cutting data from the predominant class or augmenting data from the other class.