Project 1: Comparison of ML algorithms: Decision Trees, Boosting, Neural nets, SVM and kNN

Jean-Pierre Bianchi - GT ID jbianchi3

[ ]

1 Choice of data sets

The first dataset (DS1) is the 'Wisconsin Cancer' dataset[1]. It is a binary dataset with 699 instances and 9 features of breast cells to determine if they are benign/malignant tumors. I selected this dataset because it gives 90+% accuracy right from the start, which means that, in theory, it should be not too complicated to improve, and more importantly compare which methods give the best improvements. Despite the authors claiming it had no missing data, it had some nonetheless, so I had to deal with it. There were only 16, so I decided to simply ignore those instances.

The second dataset (DS2) is the 'White wine quality' dataset[2]. It is a multiclass (10) dataset with 4900 instances and 11 features. The interesting thing here is that the classes represent a quite abstract (subjective?) criteria, ie the 'quality' of a wine. The predictors are all of chemical nature (acidity, sulphur, alcohol content, sugar etc), and I am not certain they are enough to get a perfect fit using any method since important information is ignored which for sure contributes to the perceived quality of a wine (tannins).

Worse, this dataset is highly imbalanced: the best wines (quality 9) and the worst ones (quality 3) are only a handful. Here's the class population from 3 to 9: 20, 163, 1457, 2197, 880, 175, 5. This dataset is a bit 'risky' to say the least, which is why I picked an 'easy' first dataset (which is already slightly imbalanced, 2/3 benign, 1/3 malignant), to be able to evaluate the algorithms on a more solid ground, and put the results on the second dataset in the right light. So all my analyses will be done on both sets at the same time.

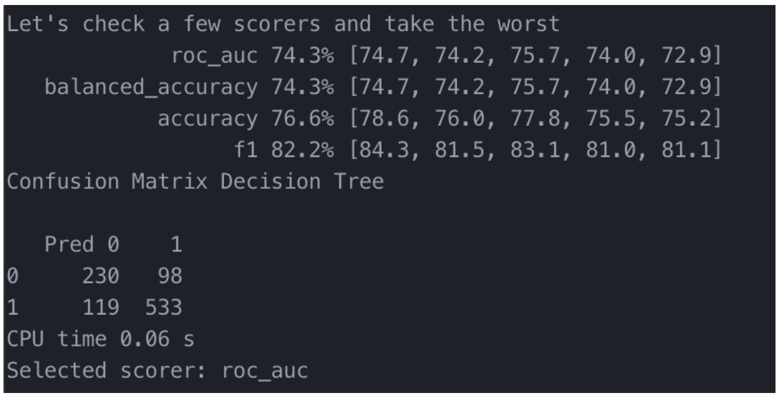
I didn't pick a dataset with categorical data because scikit-learn can't handle them, unless we use one-hot encoding which basically transforms the data set into one without categorical data anyway. Finally, as this is 'just an exercise', in the sense that we are not looking to reach the utmost perfect classification, but evaluate and compare methods, I felt that these two data sets would leave room for improvement when we get to future projects, involving feature selection and such.

1.1 A word about accuracy

An important issue, highly debated on Piazza, is the choice of an evaluation metrics, aka 'scorers' in scikit-learn vocabulary, which is only exacerbated by imbalanced datasets. For instance, precision or recall focus on one class only. F1 tries to take them both into account but ignores the overall imbalance of the dataset itself. The 'balanced_accuracy' takes into account the class population, by giving more importance to the less populated classes. But, the question remains: which one to choose?

My answer to that was using scikit-learn 'cross_validate' to evaluate a bunch of scorers right at the beginning, and pick the WORST one. Fig 1 shows what I got for decision trees and DS2. In every case, I pick the 'scoring' parameter with the worst result for all subsequent calculations. In my opinion, this already helps dealing with imbalanced data sets by picking the metrics that suffers the most from it, so an improvement with that metric can be interpreted as having 'pushed' the algorithm against ignoring the smaller class.

As you will see in the code, this is done automatically, so I won't show a figure like Fig 1 every time. I didn't select Precision or Recall since they focus on the positives or the negatives only, and F1 is based on both anyway. I also picked 'balanced_accuracy' which gives more weight to the smaller classes, and roc auc which also addresses the imbalance between classes in a different way, by considering sensitivity and specificity at various thresholds. I kept 'accuracy' as a comparison. For instance, a very different value than the others



```
Let's check a few scorers and take the worst
             roc_auc 74.3% [74.7, 74.2, 75.7, 74.0, 72.9]
   balanced_accuracy 74.3% [74.7, 74.2, 75.7, 74.0, 72.9]
            accuracy 76.6% [78.6, 76.0, 77.8, 75.5, 75.2]
                  f1 82.2% [84.3, 81.5, 83.1, 81.0, 81.1]
Confusion Matrix Decision Tree

   Pred 0    1
0     230   98
1     119  533
CPU time 0.06 s
Selected scorer: roc_auc
```

Figure 1: White wine quality scorers (example)

would reflect the an imbalance in the predictions. Anyway, since I pick the worst to work with, it means that all the others are higher, so, as we will see, a value of 74% for roc_auc with decision trees for DS2 is not bad at all.

I doubt we can go much higher than 74% for DS2 for the reason already mentioned. This article on Kaggle manages to reach 90% but they use different predictors, such as ash and phenols which are linked to taste. For the sake of this project it would have been pointless to use two data sets that classify very well.

Fig 1 also shows the confusion matrix, but I am not going to show it anymore because I feel it is not as useful as when we have many classes and we can give a quick look at the diagonal. The scorers selected are enough here.

Finally, I use the parameter 'class_weight' = 'balanced' to weight the classes proportionally to the inverse of their size, also in an effort to counter imbalance. All that being said, DS2 is still quite tricky so I decided to transform it into binary. I tried various thresholds values, and the best one is 6. In the same spirit, I have implemented a method 'equalize' which undersamples the larger class to make it equal in size to the smaller class, to have a perfectly balanced set. We will see which algorithm needs it or not.

2 Decision Trees

Below are shown various curves for both data sets. I used scikit's DecisionTreeClassifier. For these analyses and the ones further below, I have used cross validation as the first barrier against overfitting. It's quite easy to implement since scikit-learn offers a 'cv' parameter and does it automatically for us. The most severe scorer chosen by the code was F1.

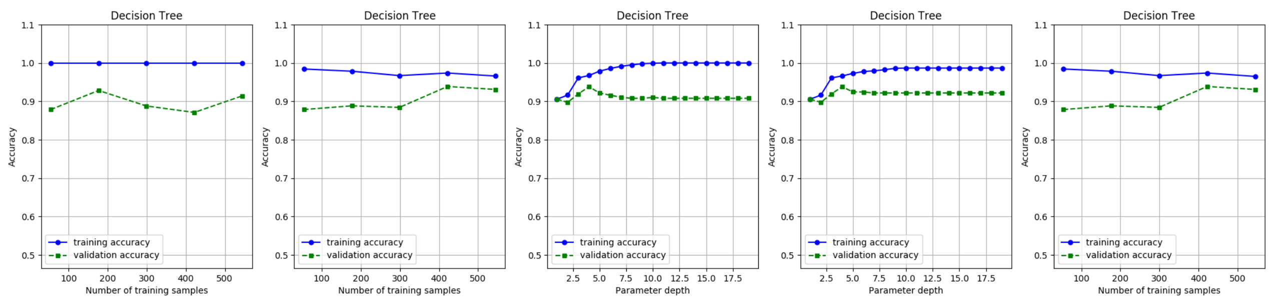


Figure 2: Wisconsin Breast Cancer Dataset

The first graph is the learning curve with scikit-learns algorithms 'raw', ie wit the default parameters. We can see that DS1 has a bit of variance left, while DS2 has very high variance (gap between both curves). Both training set performances are perfect, which means the decision tree algorithm totally overfits the data.

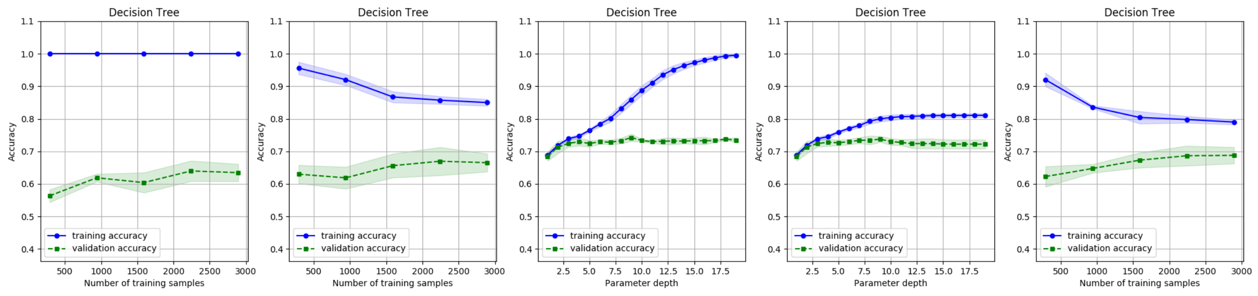


Figure 3: Wine Quality Dataset

The second graph is the learning cuves after a first round of gridsearch using max_depth and entropy vs gini. This gives me a performance of 93.8% for DS1, using entropy and 4 levels only. For DS2, 74.2%, entropy and depth 9. We can see that the variance in both has decreased, but it is still a large for DS2.

A complexity analysis (3rd graphs) shows more clearly that the variance is quite small for DS1 if we prune the tree at max_depth 4, which is good, and if we increase the depth, the algorithm starts overfitting. It looks like with a bit more data, the variance would be totally gone. But for DS2, the variance is still quite big at max_depth = 9, hence it overfits. We can see that gridsearch picked 9 because of the small bump on the validation curve, but, after a visual inspection, I would pick 4 or 5 since the accuracy there differs by 1 of 2%, but the variance is very small. So pruning the tree at depth 4 for both sets kills the variance almost totally.

I then tried gridsearch with several parameters, and couldn't really improve anything, except with 'min_samples_split' which is 2 by default. It controls the minimum number of samples required to split an internal node, or a proportion of the total instances if it's a float, so it is quite obvious it is also a method of pruning, by keeping the algorithm from creating small branches for a few instances only, which would lead to overfitting.

The gridsearch didn't improve the performance, but we can see in the 4th graphs, that it did significantly reduce the variance. If you compare graphs 3 and 4 for each data set, you can see that the variance has decreased for DS1, and very significantly for DS2, which reflects in the final learning curves (5th graph). However, this has been done at the expense of bias since both curves seem to converge far below from the ideal accuracy.

The table below gives the performance metrics of both datasets. The first column 'F1' is the F1 accuracy score of the test set before we begin optimizing, ie with the default parameters of the scikit decision tree classifier. The second 'F1' is the one that interests us the most, ie the F1 score on the test set after optimization. We can see that, despite pruning, we managed to improve the score for DS1. For DS2, as we previously explained, forcing the algorithm to take fewer levels than found by gridsearch resulted in a small loss (2%), despite the variance that we could not eliminate totally.

	F1	F1	Training Time (ms)	Fitting Time
DS1	89.7%	96.5%	4.8	1.5
DS2	74.4%	72.3%	12	1.3

Table 1: Decision Tree performance metrics

3 Boosting

From hereon, I'll use scikit's pipelines which are very convenient for applying various methods serially on a data set. I think it's a good practice to normalize the data with StandardScaler (unless we're dealing with sparse data...). Next in the pipeline is scikit's AdaBoostClassifier using the optimized Decision Tree from above. However, I increased the pruning by lowering max_depth to 3 to test Boosting's abilities. Going further down didn't produce better results. The graphs are below, in Fig 4 and Fig 5.

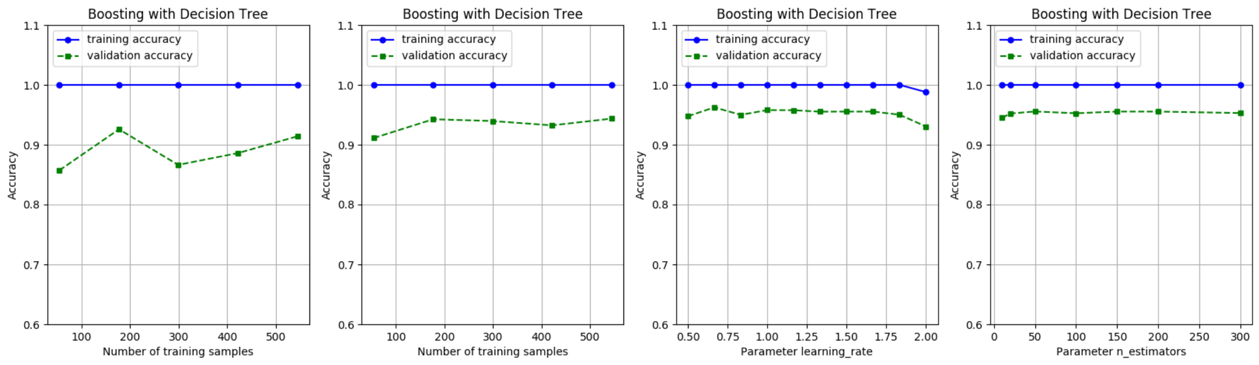


Figure 4: Wisconsin Breast Cancer Dataset

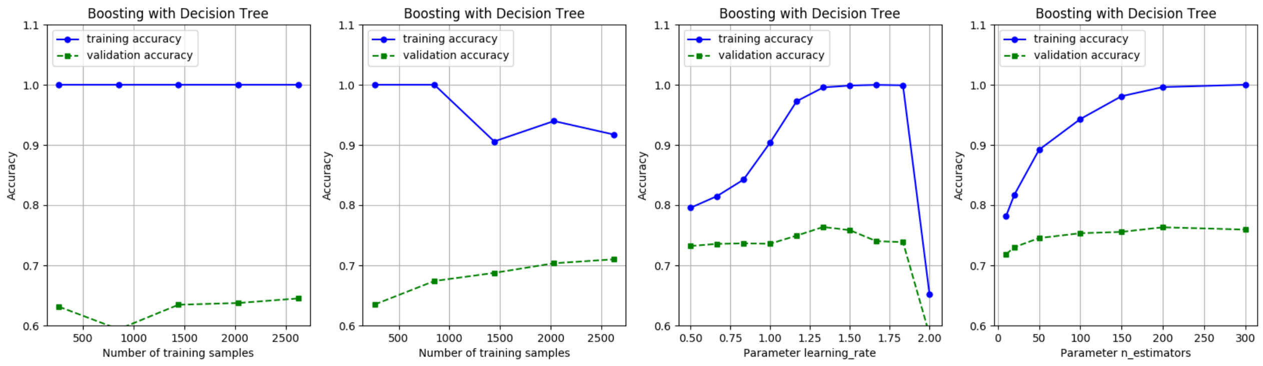


Figure 5: Wine Quality Dataset

Despite managing to improve the final scores (see Table 2), it was a bit frustrating because Boosting does not have many parameters to play with. Two to be precise: the number of stumps (n_estimators) and the learning rate, which, on top of it, are not totally independent.

The first graphs are the learning curves with the default algorithm from the library, except that I reduced the n_ estimators to two since I think the default value of 50 is already too high for this problem. Gridsearch found optimal parameters and improved the score. The learning curves improved as well, the variance has been reduced, although I was expecting more. Especially for DS2 since I used the exact hyperparameters as in Fig 3. The next 2 graphs are complexity curves for learning_rate and n_estimators. If you look at the code, you have to know that I always, meaning for every method, do several gridsearches to zero-in on the ranges I eventually use for the graphs.

The optimal learning rates were 1.7 for DS1 and 1.45 for DS2. The complexity curves confirm we can't keep increasing this parameter because it's the rate at which the contribution of each classifier shrinks. We get warnings if we try because, I guess, the algorithm can't finish implementing the required number of stumps, unless we decrease it, hence the trade-off.

The last graph shows the complexity curves for the number of estimators, and there too, I couldn't find any good idea to reduce the variance. We can see that there is no point in increasing the nb of stumps in the hope to get a much higher accuracy.

	F1	F1	Training Time (ms)	Fitting Time
DS1	89.7%	97.6%	1630	118
DS2	72.6%	78.4%	2400	72

Table 2: Boosting with Decision Tree performance metrics with 500 stumps

The metrics are given in Table 2. It is once again the F1 scores calculated on the test set, before and after optimization. We can see that we have nonetheless managed to improve compared to Decision Tree results in Table 1. For DS1, 97.6% actually means we have 1 false positive and 1 false negative, so that is going to be hard to beat in my opinion.

I also tried to fine-tune the result by modifying the parameters of the base estimator, ie ccp_alpha, and I reached 82.3% for DS2, but I think it was just luck (aka variance) because it was not supported by the values reached by the validation curves. Let's see if we can reach it or beat it with SVM. Before we move on, let's notice that the training and fitting times have exploded compared to a simple Decision tree, due to the high number of stumps. These are not considered 'iterations' so I didn't run a time analysis.

4 Support-Vector Machine

I used scikit's SVM to generate the 4 figures below. The code picked F1 once again so we'll go with it. I chose the linear kernel because it's the 'basic' one and I thought it would be enough for DS1, which is right. Then I chose a more complex one, rbf, with the hope it could improve on previous results by managing to map some hidden complex structure in the data in an indefinite dimensional space. Using such a complex kernel is the first chance I got to try to deal with this difficult DS2 because the previous methods are basically 'chop/split the data and do it again'.

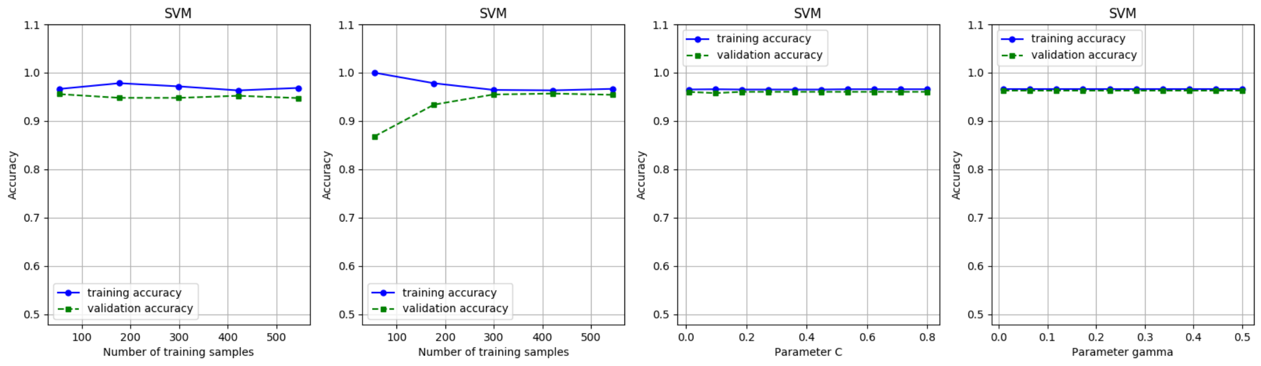


Figure 6: Wisconsin Breast Cancer data set - SVM with linear kernel

In Fig 6, the first graph, as always is the Learning curve with the default parameters, and the second after optimization with gridsearch. As always too, to avoid doing very large searches, I did a few couple of gridsearches manually to zero-in on the right parameter ranges before doing a final much finer search. It is a surprise to see how both curves in learning curves are very close together right off the bat. However, there is a small bias in return. And after optimization, both curves are very close to each other, which is the first time we see such a low variance.

Then I plotted the complexity curves for C and γ. Modifying C doesn't seem to change the variance, while, in that regard, γ should be kept low. I think this algorithm is too powerful for this data set, although there's a

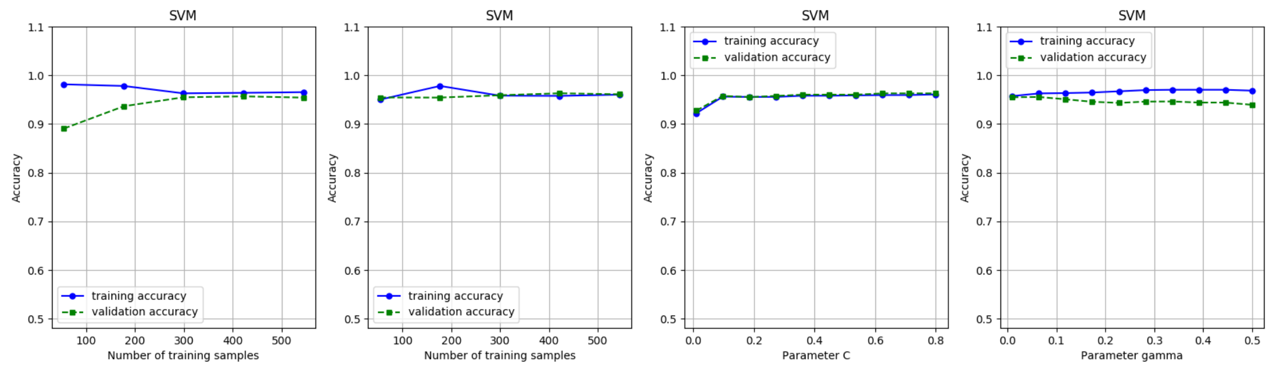


Figure 7: Wisconsin Breast Cancer data set - SVM with rbf kernel

handful of instances that refuse to classify properly. We'll see later, but I'm now thinking those are atypical cases or errors. The analysis is basically the same for the rbf kernel in Fig 7.

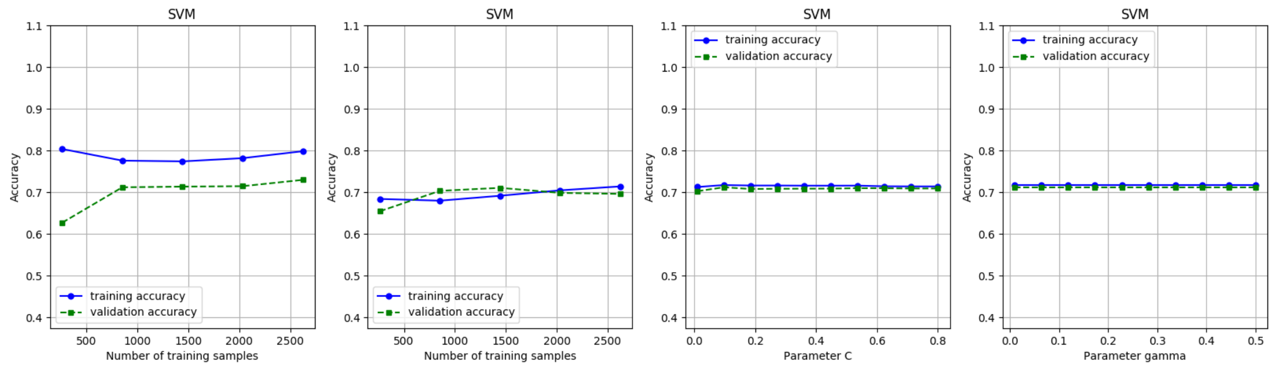


Figure 8: Wine quality data set - SVM with linear kernel

Now, for DS2, things didn't go as hoped. We still have this 75-80% F1 accuracy that I can't break as one can see in Fig 8 and Fig 9. These curves are very similar to the ones of DS1 except that they are 20% below. The linear kernel manages to reduce the variance quite well, but we have too much bias for it to matter. In fact, for the linear kernel, the performance even dropped as you can see in Table 3.

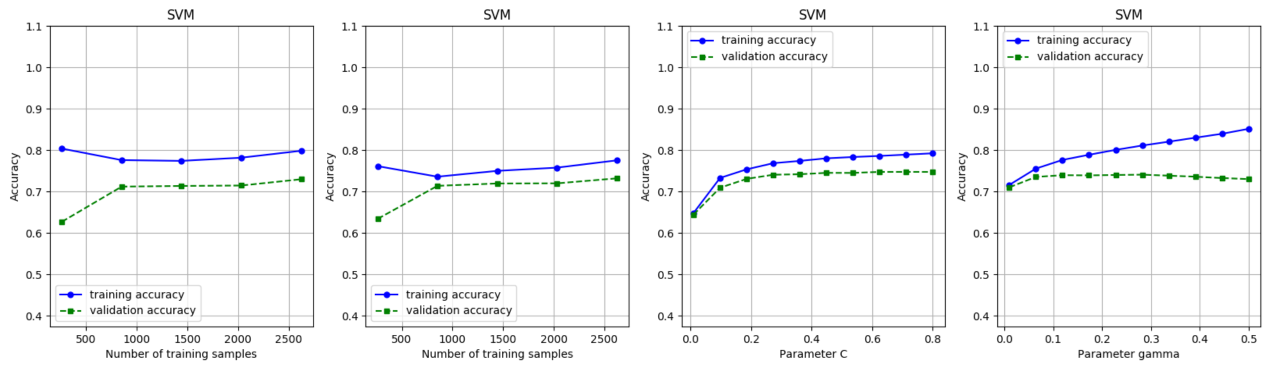


Figure 9: Wine quality data set - SVM with rbf kernel

To finish with SVM, it's impossible to miss how the variance has been dramatically reduced in all 4 cases, compared to the Wild West at times with the previous methods. I think it is intrinsically due to SVM's

	Kernel	F1	F1	Training time (ms)	Test time (ms)	C	γ
DS1	rbf	96.4%	95.2%	43.9	12.8	0.785	0.02
	linear	96.4%	94.0%	9.7	1.7	1.62	0.01
DS2	rbf	76%	78%	284	48	0.316	0.1
	linear	76%	70%	223	21	0.09	0.01

Table 3: SVM metrics

capability to map and split data in other dimensions. I see it in my mind as the data points being able to move (over different data sets) as long as they stay in 'their own' dimensions.

5 k-Nearest Neighbors

The results are shown below in Fig 10 and Fig 11. I used scikit KNeighborsClassifier. The code once again chose F1. There are not many parameters to tweak with kNN. Besides the obvious 'k', ie n_neighbors, I also searched on weights and metric as you'll see in the code.

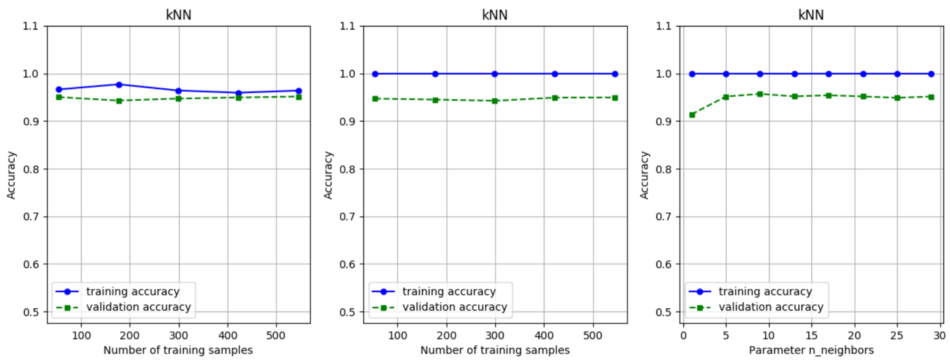


Figure 10: Wisconsin Breast Cancer data set

I was surprised by the big increase in variance, especially for DS2. The math formula for the variance decreases when k increases. I tried but it did not produce any noticeable effect, in the result or in the curves. Even in DS1, which has been pretyy good before and after optimization, we can see in the second graph in 10 that the variance takes a small hit, which is why F1 didn't improve in Table 4. The complexity curves for DS1 show that k has barely any influence on the variance, unless it goes under 5.

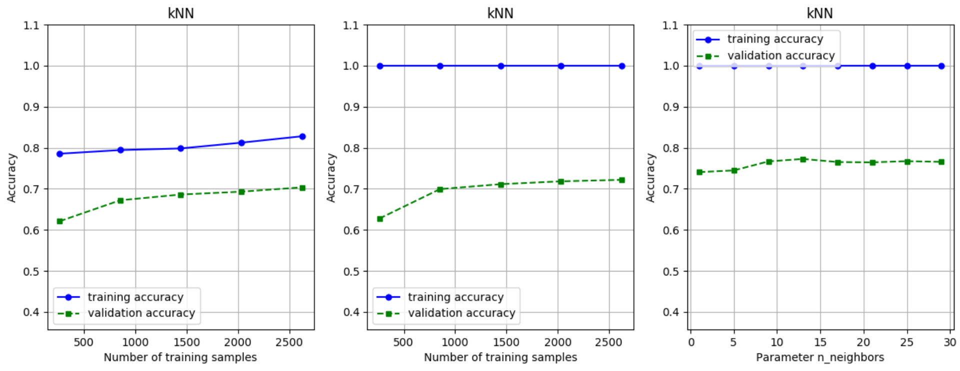


Figure 11: Wine quality data set

	F1	F1	Training time (ms)	Test time (ms)	k
DS1	95.1%	95.1%	6.0	5.1	9
DS2	74.6%	81%	8.1	64	13

Table 4: kNN metrics

For DS2, kNN has increased the variance tremendously. The complexity curve indicates we could try to increase k, but as I already explained, it had no effort on the score. And that is where the surprise is, because with F1 = 80%, it's our biggest score so far (except some fluke results I had with SVM). I changed the seed several times, but the score remained 80% $\pm$ 0.5. So much for high variance !? Although I must say I was expecting a good performance of kNN on this data set because, in my mind, when the algorithm looks for the k nearest datapoints, it doesn't matter if there is a thousand datapoints far away in another class, so I can see kNN good at classifying even small classes. I think some of this played a part with this difficult data set (even if I balanced it, what I said was about the features).

6 Neural Networks

I used scikit MLPCClassifier method. The code picked F1 as the most severe scorer, for both sets, again. From Fig 12 and Fig 13, we can see from the first graphs that the neural networks already have a low variance with the default parameters, on which it did only marginally improve after that, as the second graphs show.

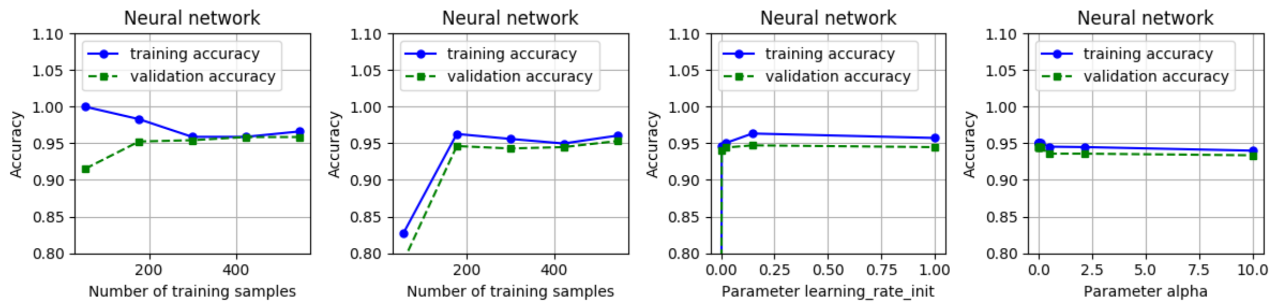


Figure 12: Wisconsin Breast Cancer data set

As you'll see in the code, I scanned a wide range of parameters (type of activation, solver, learning rate), plus various hidden layer depths, alpha ranges, learning_range_init ranges to optimize the result, with barely no sizeable improvements (2nd graphs): same bias, same variance.

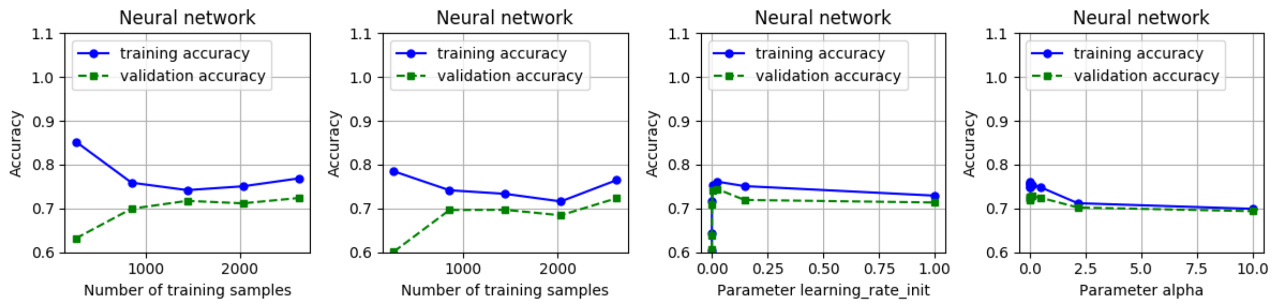


Figure 13: Wine quality data set

For both data sets, I did a few complexity analyses, but as we can see in graphs 3 and 4, in Fig 12 and 13, both the 'learning_rate_init' and 'alpha' must be quite small, as the gridsearch found out. The 'relu' activation was always picked, as well as the 'adam' solver. I ran out of ideas on how to improve these results. The next project will hopefully shine some light on it.

	F1	F1	Training time (ms)	Test time (ms)	layers
DS1	94.0 %	96.5%	104.0	2.1	25
DS2	74.6%	76.3%	281.1	2.0	12

Table 5: NN metrics

The results are summed up in Table 5. We can see how NN 'started high', even with a 3 hidden layers, and didn't make any significant improvements.

Finally, since this algorithm uses iterations, here is the timing curve for different data sizes. I took the biggest data set, ie DS2, to make it more challenging to the neural net. The time is the time needed for a neural net with 9 layers to fit the data. As expected, the time increases with the size of the data. The spikes are probably due to the random process selecting the data. If I had more time, I would have done it, say, 10 times, for each size, to smooth the curve.

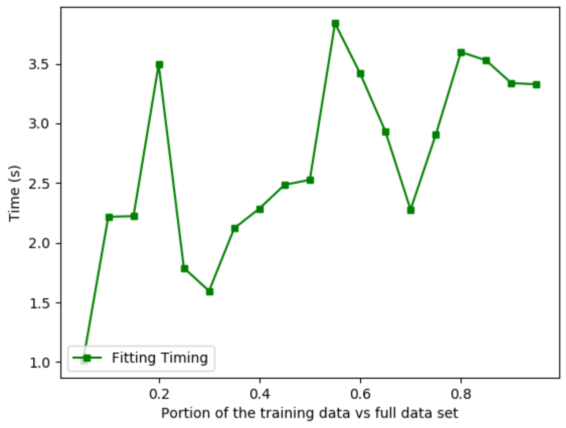


Figure 14: NN timing curve on Wine quality data set

6.1 Comparisons and conclusions

First of all, these two data sets lived up to their promises. DS1 showed how ML algorithms can really model real phenomenons, while DS2 showed how difficult it can be to do so. An F1 80% accuray is not bad at all, but it kept me asking for more. For sure, they allowed to show the strengths and adaptability of these algorithms without doing fancy stuff like features selection.

The table below sums up all our best results. We can see how the simplicity turns into very quick training and test times, while giving the worst performance. Which boosting improves quite a lot (best performance for DS1 and second best for DS2) at the cost of a huge training time, which, of course, depends on how many stumps we decide to use. We don't see in these numbers, but SVM had an astonishing variance reduction at the cost of training times. kNN actually shone with a very high performance for DS1 and the best for DS2, for an extremely small training time and acceptable test time. Neural nets with 3rd and 4th in terms of performance for DS1 and DS2 respectively, while the training times were quite high. So, for sure, not the first algorithm to try.

	F1	F1	Training time (ms)	Test time (ms)	Algorithm
DS1	89.7%	96.5%	4.8	1.5	DT
	89.7%	97.6%	1630	118	BOOST
	96.4%	95.2%	43.9	12.8	SVM-RBF
	95.1%	95.1%	6.0	5.1	kNN
	94.0%	96.5%	104.0	2.1	NN
DS2	74.4%	72.3%	12	1.3	DT
	72.6%	78.4%	2400	72	BOOST
	76.0%	78.0%	284	48	SVM-RBF
	74.6%	81.0%	8.1	64	kNN
	74.6%	76.3%	281.1	2.0	NN

Table 6: All metrics

7 References

[1] W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. IST/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, San Jose, CA, 1993.

[2] P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties In Decision Support Systems, Elsevier, 47(4):547-553, 2009.