Machine Learning

Alexandre Castelnau

Fall 2021

Table des matières

1 Introduction - Description of the problems

I decided to take two datasets to be able to make this comparative study of the five supervised machine learning algorithms : the first one on hospital mortality, the second one on BMW cars. All the datasets have been retrieved in CSV format and are put in a Pandas DataFrame.

1.1 Hospital mortalily

The hospital mortality dataset includes 1177 tuples. Each of these tuples compiles dozens of attributes on each patient (age, gender, BMI, diabetes...). Later on, we will not have to process all the attributes initially present. Among them, the binary attribute "outcome" represents the data that we will have to predict. "Outcome" is 0 if the patient came out of the hospital alive and 1 if he died. It appears therefore that this first problem is a pure binary classification problem. Meanwhile, even if the problem may seem very simple, it becomes harder when we think about the relative small size of the dataset and the fact that high predictive attributes (showing high value of correlation with the *Outcome* attribute) don't exist here (unlike the following problem). The following figures show the first data points and the correlation matrix.

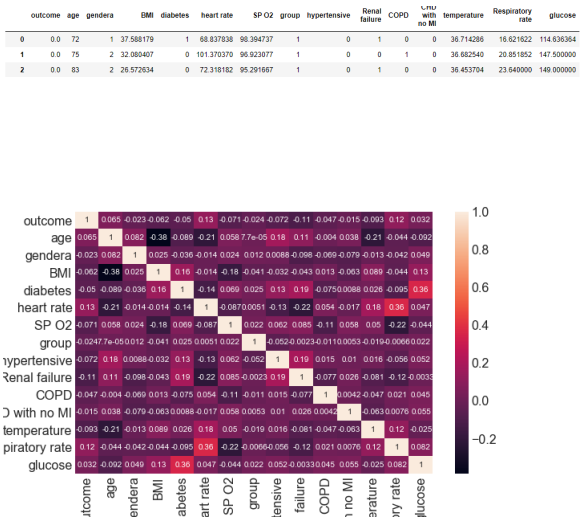

FIGURE 1 – Hospital mortality dataset

1.2 BMW cars

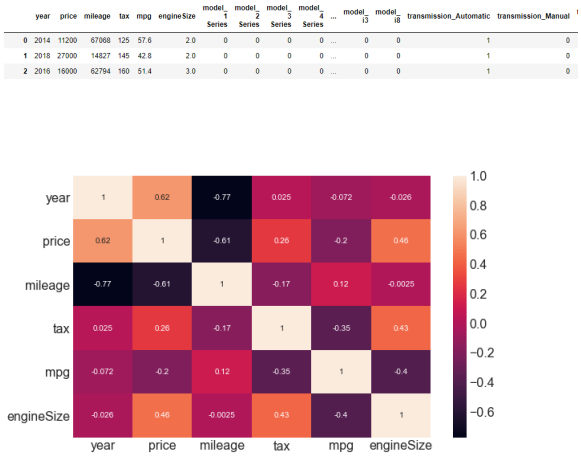

FIGURE 2 – BMW cars dataset

The BWM car dataset contains 10781 tuples. For each of the training points, we find the mileage, the model, the year of sale on the market, the engine size... The purpose of all these attributes is to be able to predict the "Price" attribute which is the resale price of the vehicle in question. Unlike the first problem, this is a regression problem (and not a classification problem). If I had been asked to choose two classification problems, I made such a choice for more diversity and as Dr.Isbell said, one can consider any problem using a finite number of data as a classification problem. As we can, this dataset can easily provides insights for the future construction of our models. Indeed, the correlation between price and some other attributes is very high, this tends to facilitate the construction of predictive models.

1.3 Model building

To build the models, I used a Jupyter notebook using most of the basic Python libraries needed for Machine Learning (*matplotlib*, *pandas*, *sklearn*, *numpy*...). These libraries will provide a great help to implement the following models : **Decision trees**, **Neural networks**, **Boosting**, **Support Vector Machines** and **k-Nearest neighbors**. Let's look how they behave with our datasets.

2 Decision Trees

2.1 Hospital mortality problem

2.1.1 Overfitting and pruning

First, I began to explore the possibilities of decision trees. Soon, the problem of overfitting appears with this model because adding depth (and so, complexity to the model) to the model is not always the best idea.

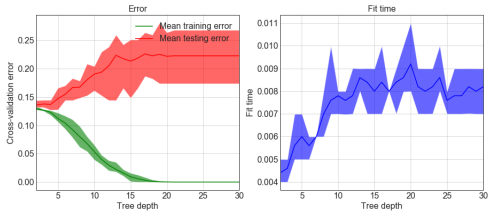

FIGURE 3 – Error and fit time / Tree depth

Indeed, the addition of branches to the model is done by wanting to gain information (entropy decrease or Gini index) on the training data set. But this can lead to a singularization of the model and a loss of generality. This is why Figure 3 shows us that the error decreases and converges to 0 for the training data as the complexity increases while the error increases for the test set. By the way, we can see that the fitting time is profoundly tied with the depth of the tree.

Pruning methods are useful to avoid overfitting, I tried to implement some of them to see if they are relevant (they should be, but let's make it sure).

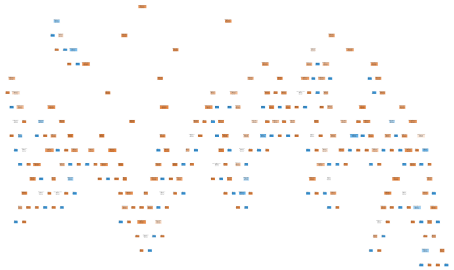

FIGURE 4 – Decision tree before pruning

FIGURE 5 – Decision tree after pruning

As we can observe (see Figures 4 and 5), pruning allows me to reduce the complexity of the model (from a *max-depth* over 20 to only 5). Moreover, this decrease in complexity is accompanied by an improvement of the classification on the test data (the decrease of the classification quality on the training data was predictable and is not the criterion determined for the model construction).

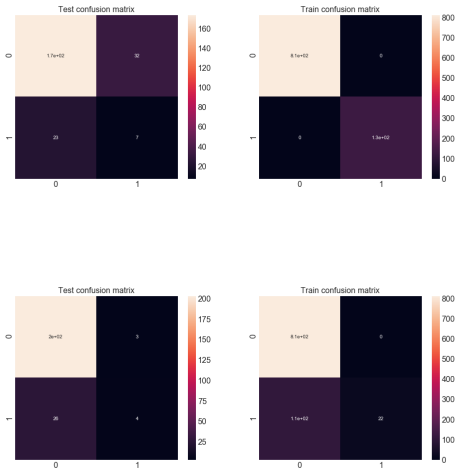

FIGURE 6 – Confusion matrices before and after pruning

Confusion matrices clearly indicate on the testing set :

$$\mathbb{P}_{Pruned}(X_{misclassified}) < \mathbb{P}_{NotPruned}(X_{misclassified})$$

But we are dealing with an unbalanced dataset, the real goal here is to determine the people who are most likely to die. I tried to play on the "*class weight*" hyperparameter of the model, to try to improve the prediction on the hard cases

(even if it means "losing" in global score) but it did not lead to any improvement. Here, we can see that the overfit tree finally detects better the most complicated cases (which are the ones we are most interested in).

2.1.2 What new data points give us ? - Learning curves

Also, we want to know what can be the effect of adding new data points on the model, see if the performance of the model improves with the addition of these new points.

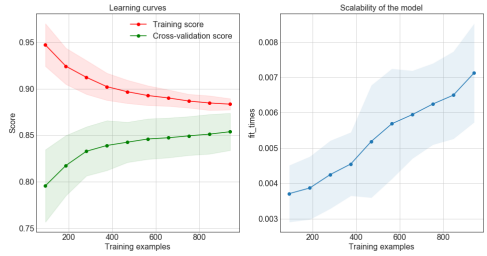

FIGURE 7 – Leaning curve (*DecisionTreeClassifier*)

Let's try to analyse Figure 7. First, we can briefly look at the second graph. It clearly indicates one other time that the scalability of the model seems to be linearly linked with the number of available points (the fit time is so linked with the number of data - Figure 7 - and the depth of the model - Figure 3).

Therefore, the first one is way more interesting. We can see that score over the training set decreases while it increases for the test set. This is quite normal, indeed, I limited the depth of the model (depth of 5 - best hyperparameter), as new data appear, the leaves of the tree will surely lose in purity since they will not be able to subdivide anymore. Indeed, when the model can contain up to 5 levels and it must, for example, classify only ten points, the leaves will be pure but when we have 1000 points to classify, the purity of the leaves will no longer be present. To make a comparison, we can think about Lagrange's interpolation theorem. We are considering only polynomials with degree under k and assume having n points where each point might have little noise on it (i.e. $\forall i \in [1, n],\ x_i = x_{i0} + \epsilon_i$). While $n < k$, Lagrange's interpolation theorem tolds us we can find a such polynomail that perfectly match the different points (no training error), but that is not what we want because it match noise. When $n > k$, finding a such polynomial might not be possible, we will use MSE minimization to find a model, it will not do a perfect match on the training points, but follows the trend. All this can explain the drop in score on the training set.

As for the improvement of the score on the test set, it can be explained in the following way. Each data is a point in a k vector space, when it "enters" the training set, it allows to refine the approximation function on its neighborhood. As we go along, under certain hypotheses (balanced data set...), the set of training points becomes dense in the k vector space and the approximation function is locally defined everywhere and so (while the pruned model permits to not fit the noise), the test score will increase. Meanwhile, this learning curve also shows that our data set is not big enough because, at the end of it, the training and test error are still in way to converge and adding good data point may allow to have a good model. Currently, the model still has an high variance, adding data will lower the variance and slightly increase the bias.

2.2 Pricing problem

2.2.1 Overfitting and pruning

Let see if we find similar concusions on the regression problem. I plotted the error of a *DecisionTreeRegressor* as a function of its depth.

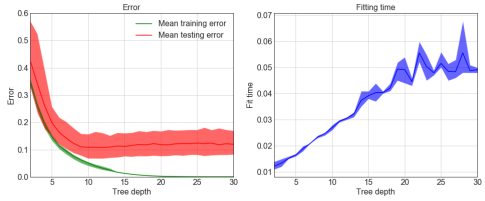

FIGURE 8 – Error and fit time / Tree depth

We can observe in Figure 8, the typical curve showing potential overfitting when the complexity of the model increases. The training error are always going down while the mean testing error decreases in a first part than slowly goes up (overfitting). This is why, I use *GridSearchCV* function, available with *sklearn* libraries to choose my hyperparameters.

2.2.2 What new data points provide us ? - Learning curves

To create the learning curves, I used the hyperparameters given by the *GridSearchCV* function.

The main one is the depth : 20. With it, I plotted those graphs.

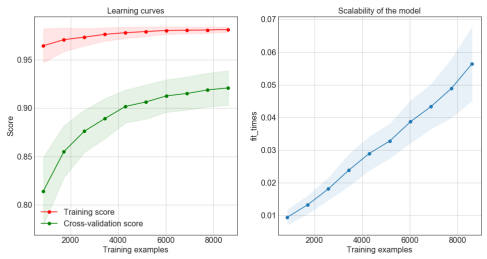

FIGURE 9 – Learning curves (*DecisionTreeRegressor*)

As in the classification problem, it seems that there is a linear relation between the scalability of the model and its depth. But, this time, both scores increase with the addition of new data points. The explanation for the increases of the cross-validation score can be the same as the one proposed in the previous part. My understanding of the slow training increase is this : if you think of this problem as a multi-class classification problem, when you have little data, the representation of the data can look incomplete. It might not reflect the representation of the points and adding new points completes the representation. However, this remains the score on a training set, so the improvement of the score remains small, but noticeable on the graph above.

Again, it appears that my data set don't have enough points for a such problem with this kind of model. At the moment, this model has a large variance. Even when we add the last point of my dataset, the accuracy, the score of the model is still increasing, we would love to add new data points to increase the accuracy of our model.

3 k-Nearest Neighbors

3.1 Hospital mortality problem

3.1.1 Overfitting and underfitting

We are now considering a *k-Nearest Neighbors Classifier*. I used cross validation over several models having different hyperparameters. The following figure shows the influence of the main hyperparameter (the number of neighbors) on the training score and the cross-validation score.

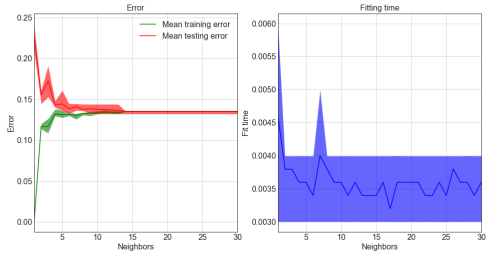

FIGURE 10 – Error and fit time / Number of neighbors

It seems that the fitting time is a constant function facing the number of neighbors considered for the model building. Also, we can see how the increase of number of neighbors taken into acount for the model building have a smoothing effect. Indeed, the prediction for a point x_0 is following this rule :

$$y_{pred} = \begin{cases} 1 \text{ if } |\{y_x = 1, x \in V_k(x_0)\}| > |\{y_x = 0, x \in V_k(x_0)\}| \\ 0 \text{ if } |\{y_x = 1, x \in V_k(x_0)\}| < |\{y_x = 0, x \in V_k(x_0)\}| \end{cases}$$

where $V_k(x_0)$ is the k-neighborhood of the point x_0, i.e. the k nearest points of x_0. When k is small (1 is the extreme case), the space is very fragmented in very small disjointed subspaces. Increasing the k subdivides the space into fewer disjoint subspaces, it softens the boundary between the different subspaces. Going to the extreme, if k is equal to the number of point available on the dataset, the prediction (and the "predictive" space associated) is always the most common class on the training dataset (it is the smoothest space possible).

In the Figure 10, we find the null error for $k = 1$ on the training test. Indeed, if we take a point x_{train} in training set, his predicted label will always the good because the neighborhood considered is the closest point to him. But, the closest will always be itself, so we obtain a null error. Meanwhile, a such model is not interesting, it overfits (perfect score on training set, but not the best on the test set). Then, a small value of k creates overfitting and like I said previously if the k is too big, it will underfit (not visible on the graph) by regression to the mean (here the most common class).

In the description of the prediction rule, we did

not talk about how we define the neighborhood. Different distances exist (it is one of the others hyperparameters available for a *k-Nearest Neighbors Classifier*). I use once again the *GridSearchCV* function to find the best hyperparameters combinaison possible (Manhattan distance and 12 neighbors). With this tuple, I split my dataset (80-20 percent for training and testing), predict outcomes with the model on the training and the testing set to creates the confusion matrix (Figure 11)

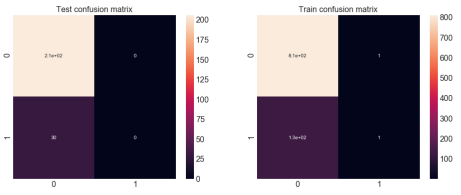

FIGURE 11 – Confusion matrix (*KNeighborsClassifier*)

It appears that this model is less accurate than the decision tree classifier we previously used.

3.1.2 Adding data - Learning Curves

Once again, we would like to see the relation between the size of data points taken into account for the model building and its performance.

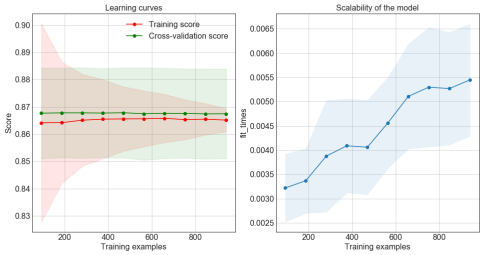

FIGURE 12 – Learning curves (*KNeighborsClassifier*)

First, it seems that the time to fit the model to the data is still linearly tied with the number of data (but a huge variance).

The first graph is somewhat intriguing. First, the cross-validation score is slightly higher than the score on the training set, but it also seems that the accuracy of the model is independent of the number of data points. Maybe the way the model divide the space in function of the number of data point quickly converge towards the final division of the space because data are well distributed. Indeed, when we add a new point, it is likely to be in the subset that predict his label (and it will not change a lot the border between different predictive subsets). Here, if I could add new points, it will not be useful, there will still have somekind of overfitting.

3.2 Pricing problem

3.2.1 Overfitting and underfitting

When we talk about the effect of the hyperparameter "*number of neighbors*" for the pricing problem, it seems that we can have the same conclusion about it : smaller values of k tend to give us a *k-Neighbors Regressor* which overfit and large values of k tend to homogenize the predictions (converging towards the mean if $k = |\{x, x \in D_{car}\}|$), it is a kind of underfitting. Also, the fitting time still appears to be independant of the hyperparameter "*number of neighbors*". Indeed, the way of the *kNeighborsRegressor* compute the prediction :

$$y_{pred} = \frac{1}{k} \sum_{x \in V_k(x_0)} y_x$$

have the same smoothing effect when k increases.

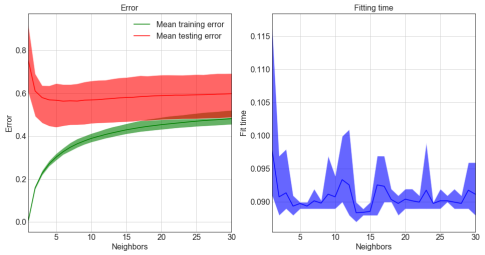

FIGURE 13 – Error and fit time / Number of neighbors

GridSearchCV gives me that the best hyperparameters are the Manhattan distance and taking 15 neighbors into account for the prediction.

3.2.2 Adding data - Learning Curves

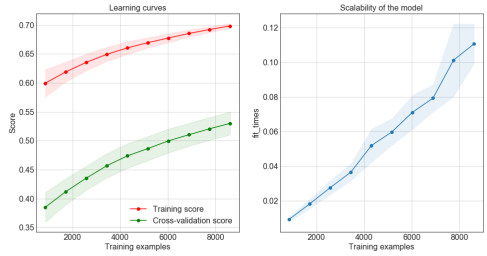

FIGURE 14 – Learning curves (*kNeighborsRegressor*)

In the pricing problem, we can see clearly the correlation between the fitting time and the number of instances over which we build the model. Also, it appears that for having good prediction results with a *kNeighborsRegressor* need a lot of data. In fact, the score is still increasing when the learning curve reach the right part of the graph. The model didn't reach his mean accuracy as quick as in the binary classification problem.

Because it seems I didn't have data enough, the results given by this model and the accuracy of it is way below the accuracy we can reach with the *DecisionTreeRegressor*. We would like to have access to a large amount of new data points to continue this slow accuracy increase.

4 Neural Networks

4.1 Classification problem

For the following model, I was not able to perform automatic search of "optimal" hyperparameters. Indeed, I tried several times to run *GridSearchCV* over different hyperparameters available for a *MLPClassifier* (maximum of iteration, number of hidden layers...) but the it runs for hours without any outcomes. So, I tried to see manually what can be the link between, for instance, the parameter "*max iterations*" and the score of the model. It seems it did not have an clear relation between (kind of independant, see Figure 15).

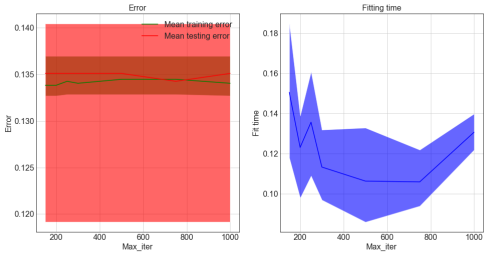

FIGURE 15 – Error and fit time / Max iterations

Meanwhile, I may have a guess to explain that. I made the plot over a range value between 300 and 1500 iterations. When I tried with lower value, an Error message appears that "*max iterations*" is not large enough to allow the model to converge. That is why the score seems independant of this paramater, we only consider converged models, they might converg the same way (if we use similar initial values).

After that, I tried to plot the learning curves for this problem.

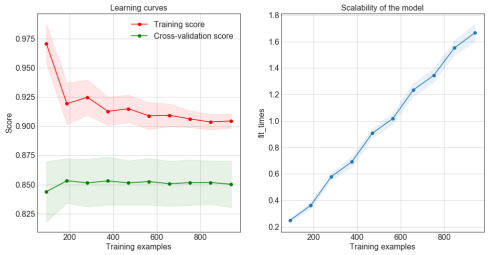

FIGURE 16 – Learning Curves (*MLPClassifier*)

We can observe more data we have, more the fitting time is important (building a Neural Network contains iterative going back and forth prediction on training set and modification of weights by back-propagation to converge towards a good set of weights). Meanwhile, adding new data for the model building does not seem to have an impact on the score (at least at some point). The cross-validation and training score don't converge, it is the epitome of a kind of overfitting.

4.2 Regression problem

During the pricing problem, I use the regressor version of a Neural Network. My first chart (see Figure 17), plotting the error and the fitting time as a function of the maximum of iterations possible. I came to same prior conclusion (the one for Classifier version). When the model has converged, adding iterations will not be useful to increase the score of the model. It is kind of logic, when we

think about it, it is the meaning of "M converge towards b" :

$$\forall \epsilon \in \mathbb{R}_+^*, \exists k \in \mathbb{N}, \forall i \in \mathbb{N} : i > k \Rightarrow |M_i - b| < \epsilon$$

For a fixed number of data points, I was able to observe that the model were able to converge with between 200 and 300 iterations.

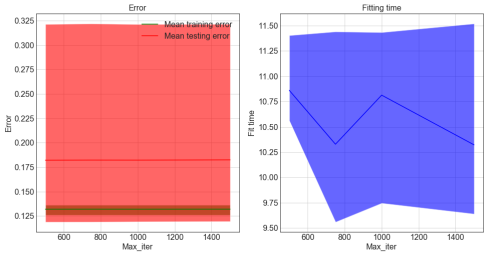

FIGURE 17 – Error and fit time / Max iterations

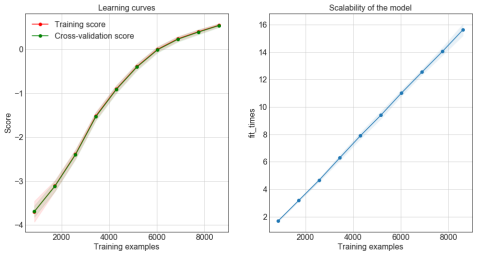

FIGURE 18 – Learning Curves (*MLPRegressor*)

Now, we can put our attention over the learning curves. *MLPRegressor* appears to be one of the most time-consuming model, this is an iterative process, when new data points add up, it cost us many time.

Also, we can observe that this model have one of the lowest score we have seen. Meanwhile, training error and cross-validation score evolve in the same way, they increase hand-in-hand. This model should not be used when faced with a weak data set. We can nevertheless note that the improvement of the scores on the training sets and the cross-validation continues while we reach the limits of our current data set. We can therefore legitimately assert that more data would be necessary to perceive the full potential of this model (which would seem to be suitable for a much larger data set than the one I have).

5 Boosting

5.1 Classification problem

I use *AdaBoostClassifier* to for the classification problem. Below, you can find the confusion matrix tied with problem for the best set of hyperparameters I could find. If we compare this confusion matrix with the one of the best model we had so far, we can observe that this model have a similar overall score but with small difference. Indeed, we face an unbalanced data set in terms of the number of representatives (many people come out alive -0- and few die -1-). The real difficulty of the problem is here is the prediction of the people at risk. Indeed, when a person enters the hospital, I could say as a "prediction" that this person will come out alive since almost everyone comes out alive. My "predictive model" (saying that everyone will stay alive) would score well, but would be useless.

Here, this model manages to make a better prediction than the simple DecisionTreeClassifier on the most complicated cases. On the 30 dying cases of the testing set the model manages to detect 1 more than the previous model. It a baby step, but still a step forward to have a better model. We know want how our model behave when new data are available.

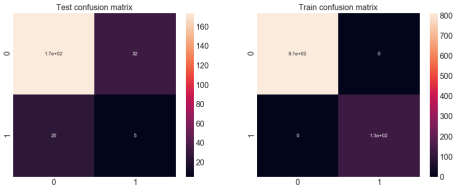

FIGURE 19 – Confusion matrix

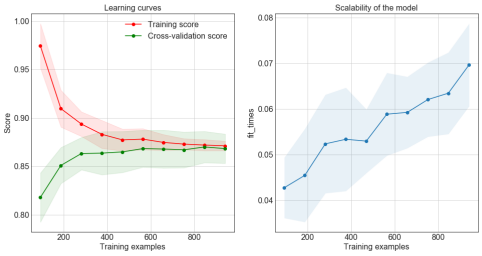

FIGURE 20 – Learning Curve (*AdaBoostClassifier*)

I was able to perform a search for the best hyperparameters for this class of model using the *GridSearchCV* function. The notable hyperparameters that I use afterwards are the following : DecisionTreeClassifier taken into account, 25 of them

If we perform a quick analysis of the Learning Curves, we can see that the AdaBoostClassifier allows to converge very quickly to a stable model. Indeed, we can see the convergence of the training and cross-validation scores on our graph. We must also remember that for this binary classification problem, we have a rather limited dataset (barely more than 1000 points). This convergence of the scores is all the more remarkable. Also we can observe that this model proposes a rather balanced variance-bias trade-off (good score, little gap).

Concerning the temporal analysis, we find once again a correlation between the number of instances and the fitting time.

Since the boosting was performed on a set of *DecisionTreeClassifier*, it seems interesting to make a quick comparison between these models.

Both models have a similar scalability (value ranging from 0.004 to 0.007 depending on the number of training points). However, the evolution of the scores is different. Even though for both models, the scores seem to converge towards the same value (close to 0.86), the speed of convergence towards this value differs. The *AdaBoostClassifier* allows to reach the stability of the model quickly (from 700/800 training values) while we can only anticipate the convergence of the *DecisionTreeClassifier* when it reaches the limits of the current dataset. *AdaBoostClassifier* thus seems much more adapted to our problem.

6 Support Vector Machines

5.2 Regression problem

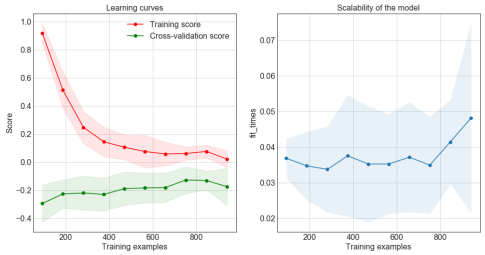

FIGURE 21 – Learning Curve (*AdaBoostRegressor - DecisionTreeRegressor base*)

About the regression problem, the results I have observed are not very conclusive. First, the convergence speed for the AdaBoostRegressor is much slower than for its Classifier version. Moreover, adding data for training has no influence on the cross-validation score besides decreasing. Moreover, we can see that the score is far from being the best of all the models we could test. We know that the score for this regression problem is the r2 score, i.e. a comparison of the prediction performance with the mean. Here, the scores seem to converge to 0, that means to be similar to a simple average. This can perhaps be explained by the final "vote" (average).

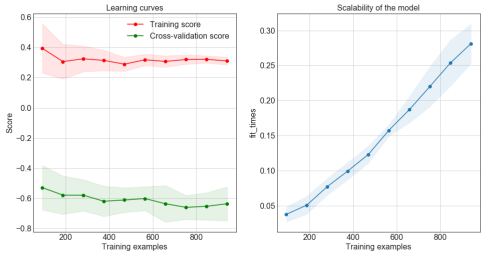

FIGURE 22 – Learning Curves (*AdaBoostRegressor - KNeighborsRegressor base*)

When I used others bases (like above with the *KNeighborsRegressor* base), the model was not better, they were even worse models.

6.1 Hospital mortality problem

I performed a GridSearchCV to find what could be the best set of hyperparameters for a SVM Classifier model. In my particular case, I found that a SVM with a polynomial kernel (highest degree of 5) seems to be the one who gives the best result. Meanwhile, when I computed the confusion matrix tied with the problem and this model, it clearly show that the SVM with polynomial kernel has a performance way under the best we have encoutered previously (it only right labeled one people who died).

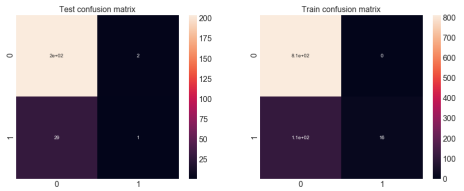

FIGURE 23 – Confusion matrix

Meanwhile, the following graph are the different Learning Curves for a SVM model. In each graph, the difference is the chosen kernel. We can observe that :

— for the polynomial kernel, adding new training points to our current dataset will not be so useful, but we can observ that the size of the dataset has a real impact on the behaviour and the performance of the model. The model, with our whole dataset seem to be well fit (no gap, no overfit) and a good score. Meanwhile, as I said previously, a good score is not necessarily a "useful model" indicator in the case of our problem which has an unbalanced dataset, where the real added value is the prediction of fatal cases. Also, it is notable to see that even if it seems to be the best kernel in term of prediction, it is the worst (by far) in term of the scalability of the model.

— for the linear kernel, the convergence is quicker than for the polynomial kernel (needs less data) and the scalability is better (it seems logic because we can think linear model as a subset of polynomial model). The score of this model is somewhat the same as the previous model.

— for the RBF and sigmoid kernel, I can observe many similarity. They both are way more scalable than the two previous SVM model (maybe that is why RBF is the default kernel) and have a similar score. But apparently adding new data does not affect the quality of the model.

By the way, I can't explain why on the three last model (all except the one with polynomial kernel), the cross-validation score is often slightly over the training score.

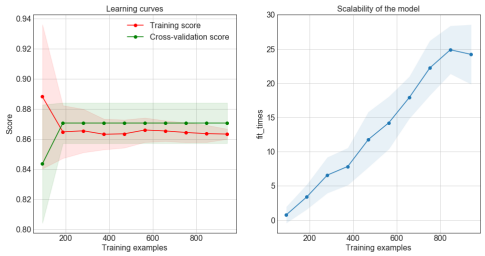

FIGURE 24 – Learning Curve (*SVC linear kernel*)

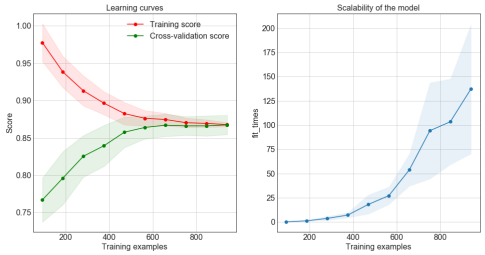

FIGURE 25 – Learning Curve (*SVC polynomial kernel*)

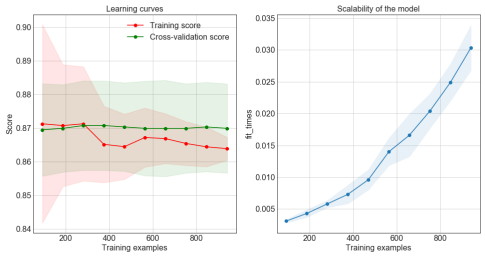

FIGURE 26 – Learning Curve (*SVC RBF kernel*)

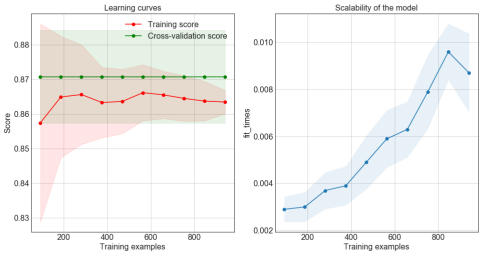

FIGURE 27 – Learning Curve (*SVC sigmoid kernel*)

With all these figures, my choice of model for my classification problem will still goes to the *AdaBoostClassifier*.

6.2 Pricing problem

As for the 'classification' version of the SVM, its 'regression' version allows to choose among different kernels. Below, we can find the Learning Curves for two of them : Sigmoid and RBF. Those models were among the worst in term of score. With my understanding of SVM, here we use soft-margins SVMs, they try to find a trade-off between large margin and fews error. The hyperparameter "C" is the one who is responsible of the trade-off (he is present in the Lagrangian or the Dual form of the problem as penality) : a large value will try to make few error and small values try to find large margin. Meanwhile, while for a classification problem SVM seems great, for regression it way more difficult to find hyperplans to divide and characterize our data. Maybe that's the reason, the performance are so bad.

FIGURE 28 – Learning Curve (*SVR RBF kernel*)

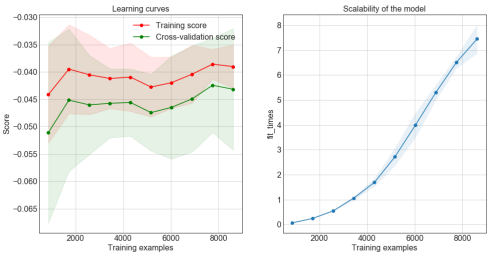

FIGURE 29 – Learning Curve (*SVR sigmoid kernel*)

7 Conclusion

What can we learn from all these implementations ?

First of all, we have observed that, faced with the same problem, different models do not necessarily have the same degree of accuracy in the prediction (as we suspected), but moreover the associated scores (or errors) can vary greatly.

Also, a lesson that emerges is the fact that the choice of its model must be based in part on the size of the data set that we have for the problem in question. We have indeed observed that the convergence of the model towards stability can sometimes require only a few hundred data points while for other models we see that tens of thousands of training points are still not enough to reach the maximum accuracy of the model.

Also the choice of the hyperparameters is important, but here is a small summary of the performances that I could observe :

- Decision Tree : tendancy to overfitting, our dataset is not big enough, difficulties to find the hard cases, more data will allow a better fit
- k-Nearest Neighbors : Don't need many data for his training, not the best model for unbalanced dataset
- Neural Networks : Do not need more data, not the best score, fit time more important, overfit of the training dataset
- Boosting : Need less data than the simple Decision Tree, our dataset is enough, performances little bit better on finding hardest cases

— Support Vector Machine : our dataset seems big enough to fit the model, according the kernel the convergence between training score and cross-validation score take more or les time but the convergence seems to be towards the same value (polynomial was the one who takes the most time)

That is what I was able to observe on the classification problem. The following observation can be made for the pricing problem :

— Decision Tree : My dataset wasn't big enough that why the model seems to overfit
— k-Nearest Neighbors : The dataset wasn't big enough for this model too
— Neural Networks : I observe that this model can be interesting for large dataset, it seems it doesn't tend to overfit
— Boosting : Bad idea, AdaBoostRegressor was not able to gives me prediction which are better then a simple mean. I would avoid it.
— Support Vector Machine :

Overall, the performance of the different models on the regression were pretty bad, but the Decision Tree was the correct in regard to the others. And for the classification problem, some model return good results (good scores) but I was disapointed (and surprised) to see that I wasn't able to increase the performance of the model on harder cases by playing with the class weights.