# Machine Learning Homework 1: Supervised Learning

Nicolas Six

February 4, 2018

## Contents



# 1 Introduction

During this report we will present the result we get for different machine learning methods. We will here explore five of them on a classification task: Decision Tree, Neural Network, Boosting, Support Vector Machine and K-Nearest Neighbors. For each of them we will study the effect of the different meta parameters on the accuracy of the classification with the help of two datasets described in the next section.

You will find in this report many plots to show the evolution of the accuracy according to one or two meta-parameters. As, many values of those meta-parameters are used, the data we have more than three dimensions and are thus hard to plot on paper. To keep a maximum of information without impacting the readability of the figures we chose the following convention: the hard line represent the median accuracy value for a given parameter value whereas, the hue area go from the minimal to the maximal value of accuracy for the given parameter value. To keep the whole report coherent the error bar looking graph will follow the same convention with the central dot representing the median value and the error bar going from minimal to maximal values.

The timing informations given in this report were measured on a desktop machine with an Intel i5-3470 CPU at 3.20 GHz with 2 cores (4 threads) and 4 Go of RAM, running a 64 bits version of the Linux 4.4 kernel. Most of the experiment were run using only one thread and were the only resource intensive program running on the machine.

# 2 Datasets

## 2.1 Credit card Fraud

The first is the famous dataset about credit card fraud in Europe. Its data or only divided in two classes: fraud and not-fraud, we also have the time and the amount of the transaction as well as 28 values coming from a PCA in an anonymization goal and, thus, do not have any human meaning. This particular version of the dataset come from Kaggle.

The difficulty of this dataset come from the sparse aspect of the data. On the 284 807 transaction only 492 are frauds. With only 0.173% of the data being frauds, an *always-not-fraud* classifier achieve very good results in terms of correct prediction ratio (99.8%).

The training dataset will have from 10 000 to 210 000 transactions to have an overview of how the training set size influence the result of the learning process. The testing dataset is composed of 236 transactions from the remaining part of the original data set, with half of them being fraud. We decided here not keep the original ratio in the test to allow a better analysis of the results and share the same metrics between the two datasets.

## 2.2 Compartmental Data - Starcraft II

The second dataset, also available on Kaggle, is composed of recorded competition game at StarCraft 2. For each game the data set provide the player name and player actions at approximate time for set of 30 different actions. The goal is then to find the player name having only the actions recorded. To allow every model to use that dataset a number of feature is precomputed on it, such as the first time of each action or

the average action per minute rate. This sum up into 72 different features for 200 class splits on 1950 games on the training set and 884 on the testing one. The training/testing set are constructed so that the same proportion are applied to each class to ensure that every class are represented on both datasets. The train dataset is then randomly flushed to allow to crop it for training and still have the possibility to have every class.

The difficulty of this dataset is in hight number of different class and the few numbers of example. 24 classes of the data set only have 1 example to train from with an average of less than 10 per class. In addition, the dataset is noisy, some players appear under more than one pseudo and so some class are very similar.

### 2.3 Normalization

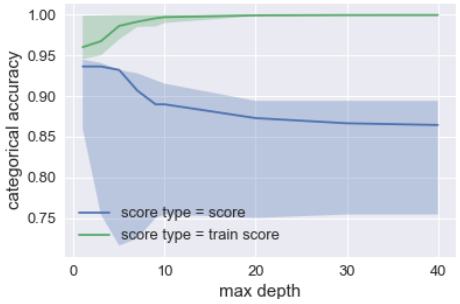
Some algorithm we are going to use with the datasets are really affected by the repartition of the values. As this problem is well-known, we didn't spend too much time on it. The solution we used to correct it is the most basic one, all the data are normalized between zero and one. More exactly the training dataset is normalized between zero and one, then the same transformation is applied on the test data set, to simulate the fact that the application data ranges are often unknown at training time.

## 3 Decision tree

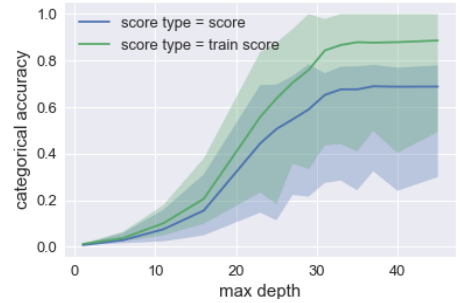
Decision trees are one of the most famous machine learning classification method. They have many advantages, among them, their really good results and the fact that the learned classification method is easy to understand by a human. We will here explore the influence of different parameters on the classification accuracy. The parameters explored during this work are the maximum depth of the tree, the min number of sample needed in a node to split it and the size of the training dataset. In this section we will explore different value of every parameter and try to understand how they affect the classification accuracy of a decision tree.

A first step in such a study will be to find the limit of our classifier on the datasets. That limit can be of two nature, overfitting or a flat level on both training and test accuracy. Those limits are both represented on Figure 1.

On 1a you can see that the decision tree quickly overfit the credit-card dataset, with a training accuracy quickly rising above 99% but an accuracy on the test set falling at the same time. It is also interesting to note that the best result are achieved for maximum depth of 1 and overfitting drastically decrease test accuracy from a depth of 3, thus, we can say that from




(a) Accuracy for the train and test dataset on credit-card




(b) Accuracy for the train and test dataset on Starcraft

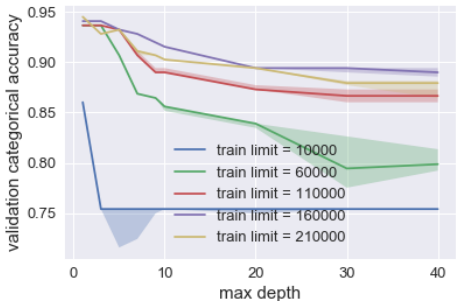
Figure 1. Evolution of the decision tree classifier accuracy according to the maximum depth of the tree

the 28 dimensions of the input space, only 1 of them is enough to distinguish fraud and not fraud transactions.

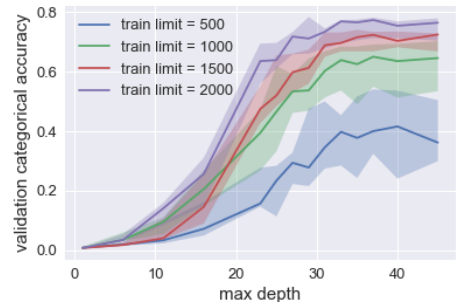
Figure 1b do not show any sign of overfitting and if it sometimes manage to get a perfect accuracy on the training data, the test accuracy stay on stable level. Showing that increasing the maximum depth of the tree further can only led to a less generalizable classifier.

Apart from the maximum depth of the tree, the parameter that affect most accuracy is the number of data used for training. The plot showing the influence of that parameter on accuracy can be seen on Figure 2. As it may have been expected, the more data the decision tree as the more accurate is its prediction.

This improvement is easy to see on Figure 2a, for the credit-card. Particularly for high depth tree, but as discussed before, those tree achieve poor performance when compared to tree with a depth less than three. Tree with a depth of one achieve very similar result when trained with more than 60 000 data. In that case an improvement still exist but as the score range from 93.5% to 94.5% it's not really significant. In addition, for those training sizes and a maximum depth of three, all the trees get exactly the same, save the tree with the full dataset that already overfit and go down to 92.8%. This tendency is then generalized on all the tree depth and so training the classifier with 160 000 data give better result than training it with 50 000 more data. This result is hard to explain, as usually




(a) Accuracy for given training dataset size on credit-card




(b) Accuracy for given training dataset size on Starcraft

Figure 2. Evolution of the decision tree classifier accuracy according to the number of training data

the more data you have the latter you overfit. This is particularly true when the train and test data are from the same source. Here the best explanation is probably that those new data introduce noise as for example a new kind of fraud that do not appear on the test set.

On the Starcraft data, the size of the training data also play a major role. The plot on Figure 2b show the same tendency to improve with the training data size, here without the overfitting problem. It's interesting to note that the evolution of the gap between each step that look like answering to an exponential law. But that comportment is mostly due to the class not represented on the training set when dealing with few randomly chosen data for training. The standard deviation of accuracy is also reduced by the augmentation number of training data, showing so that the more data you put into the classifier the less important became fine-tuning of meta parameters. The classifier will learn the best way to use that parameters and will so end up with similar result than another classifier with slightly different parameters.

The variation displayed by the hue on Figure 2 are the results of variations of the minimum number of samples in a node needed to split it. This parameter have really little importance on credit-card and only change the accuracy during overfitting. However, on Starcraft this parameter have an importance, even if smaller than on the others parameters. In that particular case, this sort of pruning only led to decreased accuracy on the test set. This is probably caused by the sometimes very few data available per class, and at the same time do not prevent overfitting as the only overfitting possible here is due to the few data available.

On an execution time point of view, the decision tree is the best performer of this report. It spends only 58 seconds to generate all the data displayed above for Starcraft. This without multi threading and using on average 85% of the thread. The full run count 208 learn and test cycle, which give us an average 279 ms to learn and classify the data. This is two times faster than KNN for example.

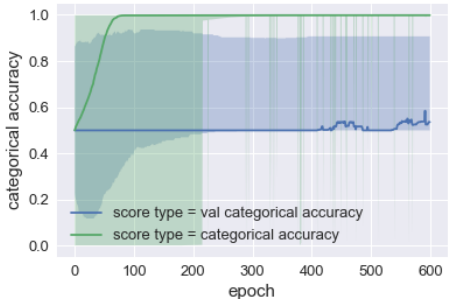
The decision tree classifier achieved here to get good results despite the problems discussed above. But the decision tree do not perform well on highly dimensional data. To counter that problem random forest were created. This classifier composed of multiple tree achieve better result than a simple one thanks to a voting process, but doing so it looses the advantage of the tree that the classification process can be understood by a human. On the Starcraft data, a basic random forest has half the error rate of the best tree of this section.

# 4 Neural Network

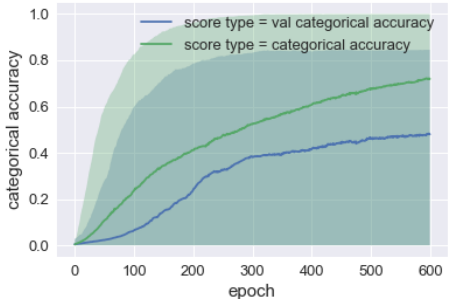
In comparison to the other algorithm the neural network is probably the one that have the most of different parameters. The parameters explored here include the number and of layers, the activation function, the use of a batch normalization between each layers, the optimizer used and the size of training dataset. Such as in other algorithm some parameters and metrics remained constant for this exploration, one of them here is the loss function. We chose to stay with a classical categorical cross-entropy as our two dataset are classification problem. Others options like adding dropout are known to improve learning stability, but are not studied here do to the physical constraints of this work.

As you can see on Figure 3, in all the parameters explored here some allowed the neural network to achieve good generalization results, but most of them end up with really poor classification accuracy even on the training dataset for Starcraft (Fig 3b). This can be explained by a number of theoretical reason, but having the data at hand we wanted to have the practical one.

The parameters that affected most accuracy is by far the batch normalization, as you can see on Figure 4a and 4b. We must admit that we were expecting batch norm to speed up training, not to change the results in such a way, particularly as the input data were already normalized. But such a comportment make sens




(a) Accuracy for the train and test dataset on credit-card



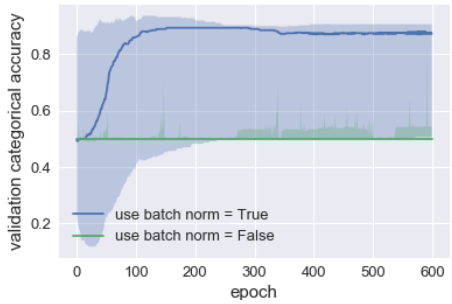

(b) Accuracy for the train and test dataset on Starcraft

Figure 3. Evolution of the neural network classifier accuracy according to the number of epoch

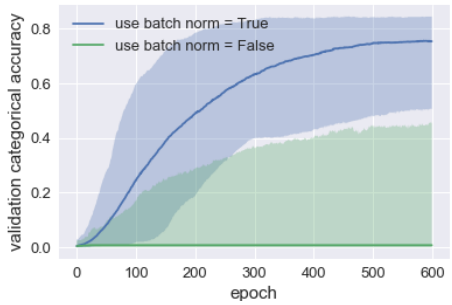
as neural network layers and the back propagation algorithm are more effective on normalized data. Figure 4c and 4d show the same plots as the one of Figure 3 but when selecting only the case where the batch normalization layers were used show that, as expected the accuracy on the training data also greatly improve with the batch normalization. From this point all the graphs will represent data from neural network with batch normalization.

After the batch normalization, the most discriminant parameters is probably the layers shape of the neural network. On Figures 5a and 5d you can see that layers have an important role and the result are dataset dependent. Thus, having the output layer directly wired to the input variable is one of the best choice for Starcraft, but the poorest for credit-card. At the same time multi-layers are bests performers on credit-card, but the poorest on Stracraft. On the first case this can be explained by the complexity of the input space regarding the small output space, were a simple weighted sum is not enough to get a good classifier. For the second one the bottleneck is on the last layer that compress the information on only ten dimension, which is not enough for the last layer to found the 200 classes accurately.

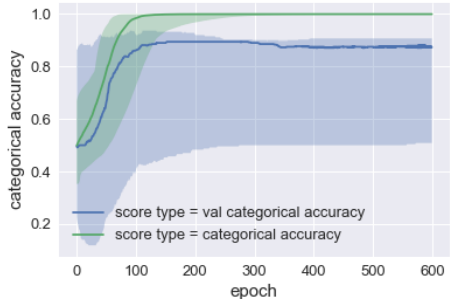
Another important aspect of the layers in a neural network is that they set the number of trainable parameters in the network. On the Starcraft case, for exam-



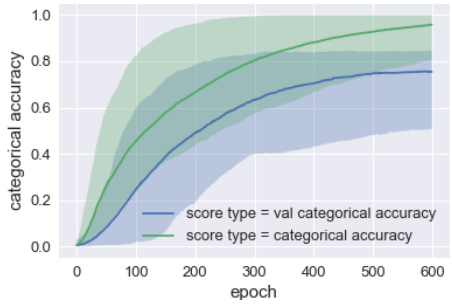

(a) Accuracy with and without batch normalization for credit-card




(b) Accuracy with and without batch normalization for Starcraft




(c) Accuracy for the train and test dataset on credit-card, with batch normalization




(d) Accuracy for the train and test dataset on Starcraft, with batch normalization

Figure 4. Neural network classifier accuracy according to the use of batch normalization

ple, the smallest number of parameters was achieved for the layer `(10)` (an intermediate layer of 10 neurones) with 3 094 trainable parameters. The biggest was for the layer `()` (no intermediate layer, output neurones directly connected to input) with 14 744 trainable parameters. A general rule in neural network is to have fewer parameters than example to prevent overfitting. With less than 2 000 training data we are far from that rule and so with a bigger number of epoch, all our neural network will have gone into overfitting. Given the small amount of data we have, trying to fit a neural network with numerous layers will have been a none sens.

The effect of the activation function is not very important. As you can see on Figures 5b and 5e, the median, max and min values are very similar for each activation function, except for the sigmoid on plot 5b which have far lower median accuracy but still have similar max values. It can be noted that a simple linear activation achieve as good results as the others. If every activation function allow similar accuracy, all of them have their particular use-case which allow to get better accuracy or shorter training time.

One other interesting fact that appear on our dataset is that the amount of data available for training do not improve the result that much. This is particularly true on Figure 5c, where training with 10 000 data get roughly the same accuracy as training with 16 times more data. The same trend can be seen on Figure 5f, where going from an average of 5 example per class to 10 only allow to improve of a 10%, mostly due to the not represented class on the training data we suppose, as they go from sixteen to zero on the same time.

Not plotted here, the optimizer chosen does not make any difference on the accuracy. We must emphasis that we only tried optimizer well suited to our classification task and do not played with parameters such as learning rate, even if we know that such parameters have a huge importance in neural network.

Regarding execution time, the perceptron is relatively slow in comparison with others machine learning methods. In addition, the learning phase of a neural network use a lot of memory. For this reason we were not able to run the experiment in the standard timing machine as the computer then need to use the swap on hard drive, which slow down the process to the point that even days of computation are not enough. For that particular measure we used an other desktop based on an 4 cores (8 threads) Intel i7-4790 CPU at 3.60 GHz with 8 Go of RAM. As the hardware is different from the others experimentation, the times cannot be compared directly. As a scale, we can say that the KNN on Starcraft take 1m 48s to complete on this machine using on average 515% of the CPU, while it take 2m 51s to run the same code on the standard machine. KNN is chosen here as a comparison because it take advantage of the multi-threading like the neural network library we use.

The run needed to compute all the data displayed on this section for Starcraft required 4h 22m and used during that time an average of 386% of the CPU. This run test a neural network for 672 different meta parameters combination, giving an average of 23 421ms to train one neural network for 600 epochs (39ms per epoch). Such time are not surprising for a neural network and are comparable to the one get for the boosting classifier, but a lot slower than decision tree for example. With the rise of deep learning a lot of speedup techniques were created in the recent years, the main one being to massively parallelize the computation on a GPU.

The perceptron tested here show correct best result on the two dataset but at the same time we showed here that neural network need a lot of fine-tuning to learn what we want them to. As in the other section a lot of parameters can be explored to complete this study, but such improvement require much more pages witch is then out of scope of this work.

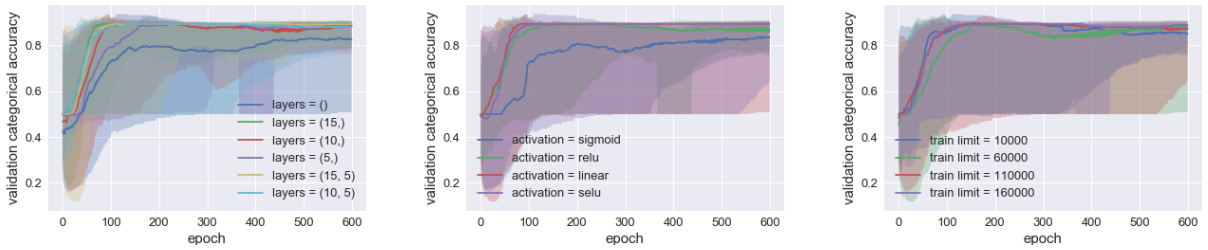
The neural network we used here, the multi-layers perceptron, is one of the oldest artificial neural network. Now most of the so called deep learning neural network are build around convolution layers among other tricks, but this kind of neural network work particularly well on image processing and does not fit on our particular problem.

# 5 Boosting

Boosting is very similar to classical decision tree, but as we will see bellow have very different results. The different parameters that we chose to explore here are the number of boosting stage to perform, the maximum depth of an individual regression estimator, the minimum number of sample needed to split a node of an estimator and the number of data used for training.

As you can see on Figure 6, the boosting classifier behave differently from one data set to the other. On credit-card, it manages to perfectly learn the training data, for every tested parameters. On Starcraft, however, even if it's still able to get a perfect accuracy on train data, some parameters restrict the boosting algorithm to more than 20% of error on the training set.

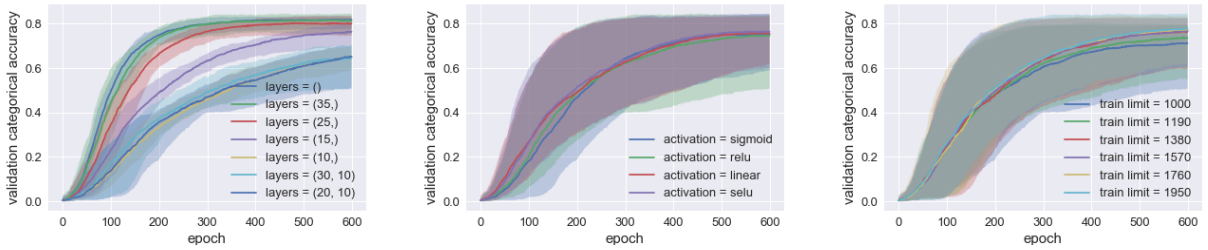
Those plots also allow us to note that on both case the maximum depth of an estimator is directly related with scores, but quickly get to a maximum and stay on it. In other word, small maximum depth do not allow to "understand" the details of the dataset whereas large value do not improve it neither. Following this observation, the others figures of this section will display data for a fixed maximum depth to get a better understanding of the evolution of the score according to other parameters. The chosen value depend on the




(a) Accuracy for different layers on credit-card

(b) Accuracy for different activations on credit-card

(c) Accuracy for different training dataset size on credit-card

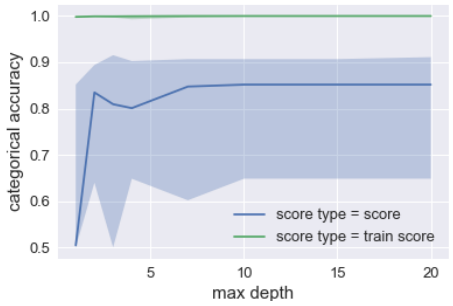



(d) Accuracy for different layers on Starcraft

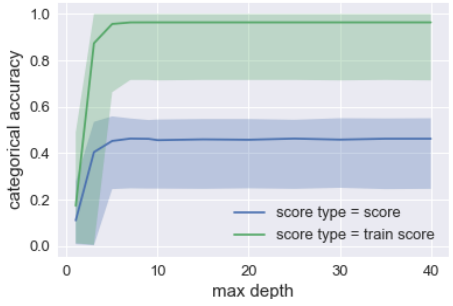
(e) Accuracy for different activations on Starcraft

(f) Accuracy for different training dataset size on Starcraft

Figure 5. Neural network classifier accuracy for different parameters




(a) Accuracy for the train and test dataset on credit-card




(b) Accuracy for the train and test dataset on Starcraft

Figure 6. Evolution of the decision tree classifier accuracy according to the maximum depth of an estimator

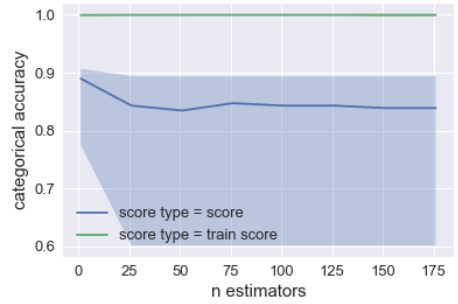
dataset and is selected amount the first values of the flat area seen on Figure 6a and 6b.

As boosting is resilient to overfit, it's also interesting to look at evolution of the scores according to the number of estimators used. Doing so we may expect to see boosting perfectly fitting on the training data given enough estimators and the test score getting closer and closer until it reach a last flat area. As you can see on Figure 7, such expectations do not fit on our data.

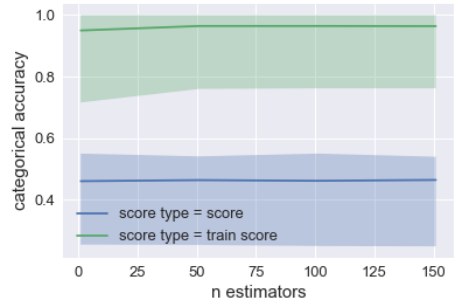
On the credit-card data, one estimator is enough to perfectly classify the training data, whether they are 10 000 or 210 000 them. In addition, trend show that the more estimator are used the more you overfit with score on test set is lower and lower, even if that overfit is not significant in comparison of the one of a neural network for example.

On the Starcraft data, the trend is similar with the difference that, as expected, boosting do not overfit. However, here as in the other dataset, the number of estimator do not really change the quality of the classification. Boosting perform here on a very similar way with 1 estimator as well as with hundreds of them. Those supplemental estimator are also not enough to completely fit to the training data.

Another interesting effect on the accuracy is produced by the minimum number of sample needed to allow an estimator to split a node. The effect of this parameter on both dataset is displayed on Figure 8. This parameter has the effect to forgive the estimator to generate rules on a too small part of the data. This parameter is also fund in decision tree to prevent over-

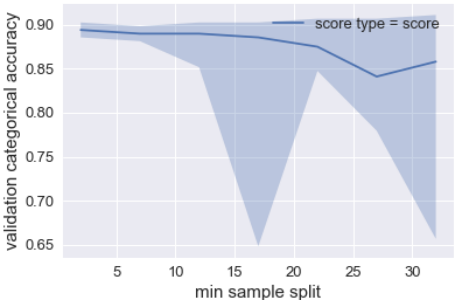



(a) Accuracy for the train and test dataset on credit-card, for $maxdepth = 5$

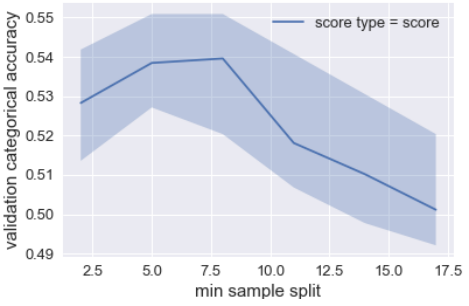



(b) Accuracy for the train and test dataset on Starcraft, for $maxdepth = 6$

Figure 7. Evolution of the boosting classifier accuracy according to the number of estimator used for a given maximum depth




(a) Accuracy for the train and test dataset on credit-card, for $trainlimit = 210000$ and $maxdepth \geq 7$




(b) Accuracy for the train and test dataset on Starcraft, for $trainlimit = 2000$ and $maxdepth \geq 7$

Figure 8. Evolution of the boosting classifier accuracy according to the minimum number of sample to split an estimator's node for a given size of training and maximum depth

fitting. Boosting is more resilient to that problem, but we will see that this parameter still play a role.

On credit-card rising that parameter value only result in lower median accuracy. That can be explained by the sparse aspect of the dataset, even with large training data number, the number of fraud stay low and in addition, we can suppose that most of the fraud are different from each other. Thus generating basics rules that highlight many fraud may be very difficult. At the same time the max value stay high and even manage to get some more percent. Meaning that a certain combination of parameters take advantage of this sort of pruning to get better generalization result.

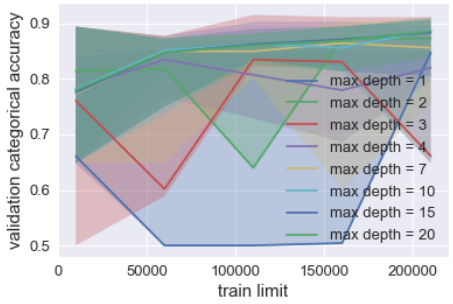
On Starcraft the behavior is different. The maximum is reached for non minimal value of the parameter, but after that apogee the accuracy keep decreasing. That last trend can be explained the same way as we did for credit-card, lot of classes have very few examples and so going above eight made the estimator ignore them, resulting in lower accuracy on test as well as on training data. The first lower point of the plot can be explained by an overfit on the training data, as stated on the dataset presentation, some different classes represent the same human player and so, are very similar. So adding rules to include one particular point may distract another class representing the same human player, and thus, decrease the generalization capability of the algorithm.

It must be noted that the variation of the accuracy on Figure 8a and 8b are very small, and are so a demonstration that boosting is resilient to overfitting.

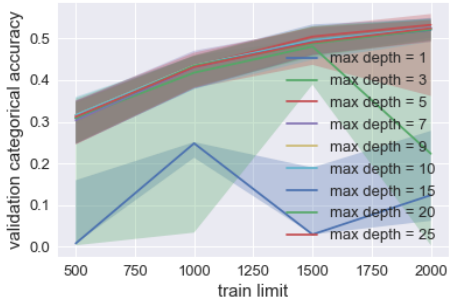
The last parameter we will study in this section is the number of data given for training. The plots of this parameter evolution are displayed on Figure 9. As general trend, we can see that boosting like every machine learning algorithm achieve better accuracy when given more data to train on. But small value of max depth are exceptions to that global trend.

On credit-card those exceptions appear for max depth value from 1 to 4. The most visible one being for a max depth of 1. For this particular value, adding new data only decrease the accuracy toward the 50% mark until it reach the full dataset length. According to how we setup the test dataset this indicate that boosting chose to classify every transaction as not fraud. Our conclusion here is that given a max depth of one boosting do not have enough degree of freedom to properly generalize on that problem. Adding new data only make it for confident in predicting the most represented class, the local maximum that appear for certain quantity of data seam to follow a random pattern. Maximum depth of 4 is on the edge and given enough data may achieve good results.

On Starcraft those exceptions show up for maximum depth of 1 and 3. Those case, as in the previous dataset, boosting do not get enough degree of freedom to properly integrate new data and end up having ran-




(a) Accuracy for the train and test dataset on credit-card, for different train max depth values




(b) Accuracy for the train and test dataset on Starcraft, for different train max depth values

Figure 9. Evolution of the boosting classifier accuracy according to the number of data used for training, for different maximum depth value

dom looking results. From a maximum depth of 7, boosting seam to have all the freedom needed and give near equal accuracy value.

Boosting is the slowest of the different method used here which still finish in a reasonable time. It took us more than 28 hours to run the parameter exploration on Starcraft. This exploration include 800 different parameters combinations, which give us an average 2 min 8 seconds per fit plus test loop using on average 99% of one CPU thread. This time is far bigger than the one of decision tree or KNN, but is still correct considering you know where to look and do not want to quantify the effect of each parameter, as we have done here.

As a conclusion on boosting, we can say that this machine learning algorithm have a really small tendency to overfit when compared to others machine learning algorithm. Regarding accuracy boosting achieve among the best results on credit-card, being around 5% less accurate than a decision tree which is about the same as a neural network. On Starcraft however, boosting get the worst result by correctly classifying about 55% of the test set. But the time needed for boosting to learn is probably its biggest drawback. Thus, we consider boosting as a good machine learning method for binary classification problem even when dealing with sparse classes, but boosting do not seam to be relevant on classification task with hundreds of classes and few examples by classes.

# 6 Support Vector Machines

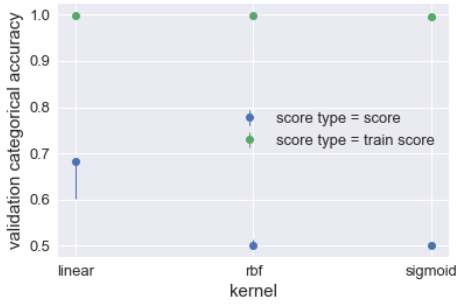
In this section we will focus our interest to Support Vector Machines (SVM) and how do they perform on our dataset for different meta-parameters values. The ones explored here, are the penalty of the error term (C), the kernel function and the amount of data provided for training. The particular version of the SVM classifier we use here manage multi-class classification problem by in a "one-against-one" approach, thus needing $n(n-1)/2$ classifier to perform the global classification, where $n$ in the number of classes.

For credit card, you will note that the size of the training set is much smaller than for other machine learning method. This difference is due to the learning time needed by SVM which is not linear with the number of data, on my machine training a linear SVM with 10 000 data take about 187 seconds, as the time complexity is more than quadratic in the number of sample going above a couple of 10 000 is discouraged by the documentation. The polynomial kernel is also absent from the credit card exploration as more than an order of magnitude longer than the linear one. Those limitations will not permit correct comparison with other methods, but even days of computation do not allow to overcome that problem.

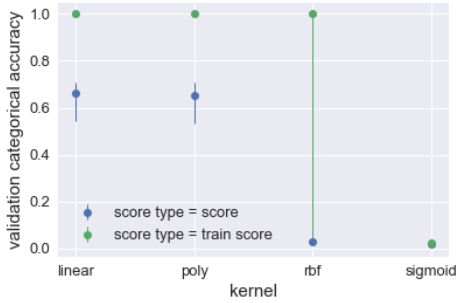
The parameter with the biggest role here is as expected the kernel used. Indeed, for SVM to be accurate the kernel function must be capable to split up the data into the different classes. Without expert domain knowledge it is really difficult to predict witch kernel will achieve good performance. It is the reason why we only explored here the default kernels proposed by the library we used. The result of that exploration is displayed on Figure 10.

On credit-card, the linear kernel is clearly above the others two, as it is the only one who manage to detect some fraud in the test dataset. Thus, we can suppose that a polynomial kernel will get at least similar results as we will discuss for Starcraft bellow.

On Starcraft, both the linear and the third degree polynomial kernel get the best results with really similar accuracy, while at the same time the sigmoid and rbf kernel get less than 3% of accuracy on test data and also on train for the sigmoid. Such a proximity on the result of the polynomial and linear kernel suggest that the underlying data are linearly separable and thus, that the polynomial functions are in fact close to linear function. In addition, the fact that the linear kernel has an accuracy 0.2% above the polynomial kernel suggest that going to higher degree polynomial will only end up overfitting the training data as they will

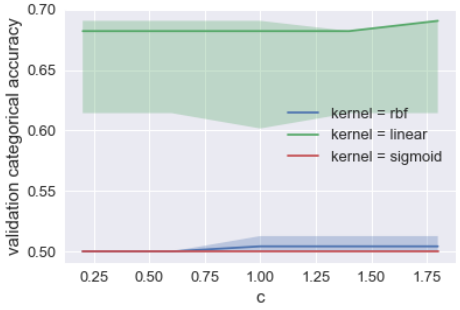



(a) Accuracy for the train and test dataset on credit-card

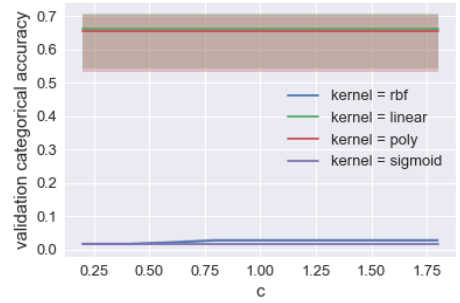



(b) Accuracy for the train and test dataset on Starcraft

Figure 10. Evolution of the SVM classifier accuracy according to the kernel used




(a) Accuracy for test dataset on credit-card, for different kernels




(b) Accuracy for test dataset on Starcraft, for different kernels

Figure 11. Evolution of the SVM classifier accuracy according to the C value

add degrees of freedom where one is already enough to classify the training data.

The penalty parameter was interesting to explore as it weight the SVM sensibility to noisy data. As always in machine learning you have to understand the data you use, to get the best results from your algorithm. Which is not our case here, so we decided to explore the surrounding of the default 1.0 value. The result of this exploration is on Figure 11.

On credit-card, we can see that C do not imply variation on the accuracy for most the three kernels. The rbf kernel get less than a 2% augmentation when C go from 0.2 to 1.8, but such an augmentation is not really significant.

On Starcraft, they are very few to no variation of the test accuracy for the different kernels, except for the rbf kernel where the accuracy increase with C around 0.7. This variation do not realy impact the results as accuracy remain around 3% as you can see on Figure 11b, but this variation also occur on the train set where in sigmoid like shape the accuracy go from zero to one around the same 0.7 value of C. Explaining the large variation of the rbf kernel accuracy seen on Figure 10b. So it appears that Starcraft dataset were not noisy in the SVM point of view and that variation of C neither improve nor worsen the accuracy, particularly for the linear and polynomial kernel.

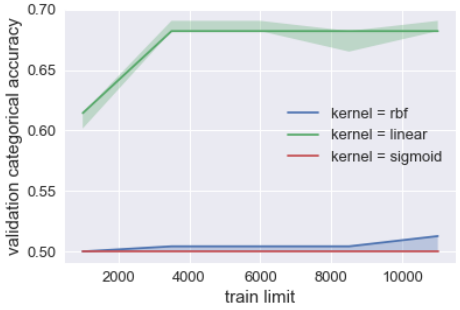
With all that in mind, the last variation displayed of Figure 11 can only come from the last meta parameter, the number data used for training.

As we already stated before, data is the heart of machine learning. But generally data as a price and so it is important to know what we could expect with more data. In the following paragraphs we will explore experimentally that field for SVM. The experimental result are represented on Figure 12.

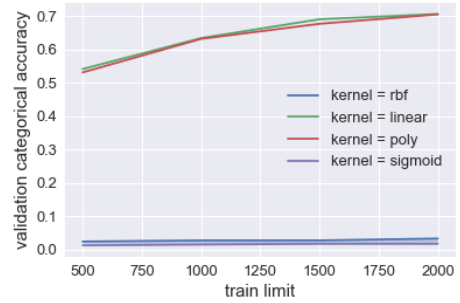
On credit-card, the linear kernel is the only one to really show improvements with the augmentation of the size of the training set, while the other two keep predicting the non-fraud class. Which give very few error on the training set, as they were only 24 fraud example for the 11 000 data in the training set.

On Starcraft, for both linear and polynomial kernel, the improvement of the accuracy with respect to the number of training sample follow a nice looking log like curve. Showing that to improve the result we may need to add a lot more data, supposing that the trend keep a log. It's also likely that adding more data will quickly bring us to a flat area. Thus, we can suppose that we are here at the limit of the model.

In both cases we can see that we are near the limit of what can be expected from a basic study using SVM. To improve further our result, the most important is probably not more data, but more domain knowledge in order to design more appropriate kernel for example.




(a) Accuracy for test dataset on credit-card, for different kernels




(b) Accuracy for test dataset on Starcraft, for different kernels

Figure 12. Evolution of the SVM classifier accuracy according to the number of training data

Regarding the execution time, SVM is the more time-consuming algorithm tested here. On Starcraft, it generated the data for this report in 3 min 37 sec for a total of 144 train test loops, giving an average of 1 507 ms per loop. It's only about three times slower than KNN and look small in comparison to neural network or boosting, but SVM execution time really depend on the data. As stated at the beginning of this section, running SVM on the full credit card dataset may takes weeks to complete and was, thus, not timed. Such a difference between one dataset to the other was not seen in any other algorithm used for this report.

In conclusion, SVM do not get the best accuracy on our dataset in comparison to other machine learning methods. It must also be stressed that regarding learning time they are far from being the best performer neither, in our experiments they were the longer to run, particularly when dealing with numerous training example for a few classes as in our credit-card dataset.

# 7 K-nearest neighbors

KNN is probably the most basic algorithm tested here. But here basic do not mean that it didn't work or is useless. As an instance-based learning, KNN do not have many parameters to play with. The main parameters we will use during this Section are the number of neighbors used to classify a new point, the p-norm used to compute the distance and obviously the number of example seen during training.

As with KNN the accuracy on the training data is 100% by design, this data will never appear on the plots of this section.

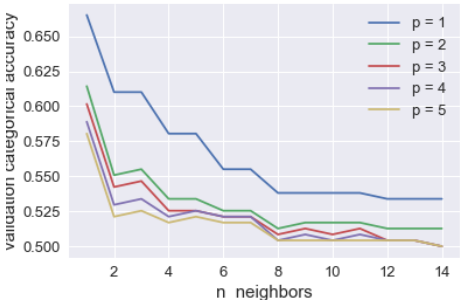
The result of the variation of those parameters on the two datasets are displayed on Figure 13. At the first glance you can see that the norm used for classification is really important. In our case, $p = 1$ allow us to have the best classification. This is not a surprise as higher norm are generally bad to measure distance on highly dimensional spaces, such as the two we have here.

On Figures 13a and 13b you can see that taking only the first neighbors allow us to get the best result on both cases. This was easily predictable as both dataset have classes with very few examples. Thus, increasing the research area only allow us to find other classes and tend to chose the more represented class in the dataset. We usually increase the research perimeter to be more resilient to noise but here the data do not allow us such a liberty. This is particularly true in the credit card dataset where the fraud are really close to legal transactions.

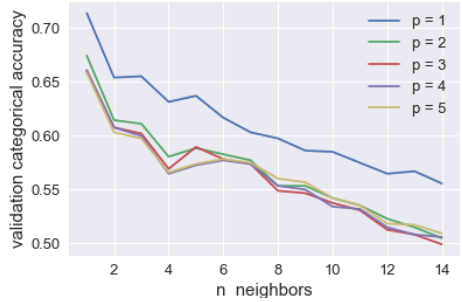
Figure 13d and 13c show the evolution of classification accuracy with the quantity of data used for training. As expected the quality of the classification improve with the addition of new training example and do not seam to overfit. However, on Figure 13c you can see a flat area in the center, where adding new point don't really improve the accuracy but neither deteriorate it. We suspect that the points added during that period are nearly identical to previously learned data.

Regarding execution time, KNN achieve correct performance. On Starcraft it takes only 2m 51s to run 280 times the learn plus test process with the parameters explored above. It must be highlighted that KNN us multiple thread to perform the classification task, with on average 295% of the CPU were used (100% being full usage of one of the four threads of the CPU). At the end on average the training of the KNN plus it execution once take 611 ms using three threads.

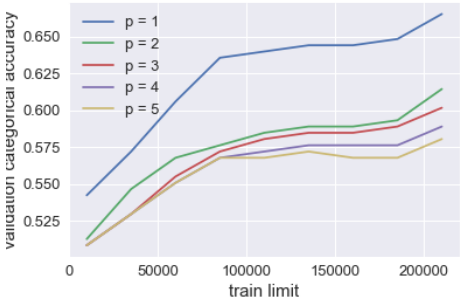
The parameters explored here are the basic one. Other parameters exist in KNN such as the norm, here we only studied the effect of different p-norm but mathematically we can define an infinity of norm, all of them with very different property. Thus, it was impossible to explore correctly that domain. But the chose of a correct norm is crucial in KNN and so the best is to fund the norm already used on that domain with the help of an expert. The same problem occur when weighting the classes, it is possible to try different weight ratio, but time-consuming and can be in itself a machine learning problem.



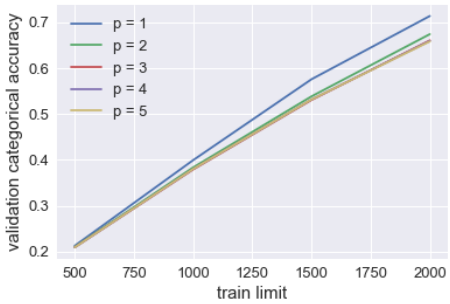

(a) Evolution of accuracy according to the number of neighbors used for the classification on the credit-card fraud dataset, with $train\_limit = 90000$




(b) Evolution of accuracy according to the number of neighbors used for the classification on the Starcraft dataset, with $train\_limit = 2000$




(c) Evolution of accuracy according to the number of training examples on the credit-card fraud dataset, with $n\_neighbors = 1$




(d) Evolution of accuracy according to the number of training examples on the Starcraft dataset, with $n\_neighbors = 1$

Figure 13. Evolution of the KNN classifier on the test set with different parameters

On the two dataset used here, KNN perform poorly in comparison with other machine learning algorithm. Some parameters may allow getting better results on the two dataset, as for example a new norm specially designed for our case. But such optimization may also tend to overfit on the train and test dataset as we will choose the norm giving the best result on the test data, with the hope to get something that can be generalized.

# 8 Conclusion

In a result point of view we can summarize by saying that on the meta parameters tried here the best performer are the decision tree for credit card and the neural network for Starcraft. In each case the second best was the other one. Those results may explain why this two methods are the most common in machine learning. But other machine learning method exist and may achieve better results, as random forest for example.

From all the machine learning method studied here, we can say that every parameter as its interest, but most of them are hard to determine without domain knowledge. Testing all of them or doing machine learning to find good values may look like solutions to this problem, but this is time and computationally consuming and even in some case impossible as when dealing with function parameters, having thus an infinity of possible values which is impossible to overcome without a very large amount of training data.

Data is really the heart of machine learning, making a computer generalize from a very few amounts of data is really hard and is the subject of multiple research around the world. The best way to a good learning is so far to have a lot of labeled examples, no matter which learning method you use.

All the machine learning methods are not equal on the land of data, but none of them is neither really different from the others. The biggest difference is in the computational complexity of their execution. For example, some results presented here have been computed in minutes and others in hours, if not in days. Thus, as always machine learning is a compromise between data, time and computational power.

In conclusion, given the same input data every machine learning algorithm may give really different results. The difference is even more visible inside a particular method, where sometimes accuracy can go from zero to nearly one with only some meta parameters change. Thus, if the name say that the machine is learning by itself, the reality is more that we have to learn how to use the machine to allow it to learn properly. This was to my mind the goal of this homework, or at least the goal I fixed to myself while doing it.