REPORT
TP SUPERVISED LEARNING

November 27, 2019

Realised by TRAN Trong Hieu & TRAN Le Minh
Group A1 - 5SDBD
Project link : https://github.com/kuro10/Apprentissage.git

I. Introduction

In this practical work, we work with dataset of MNIST-original containing handwritten digits ranging from 0 to 9. It is indeed a multiclass classification problem of 10 classes.

Our task is to try out three differents basic machine learning methods which are k-Nearest Neighbors (kNN), Artificial Neural Network (ANN) or Multi Layer Perceptron (MLP) and Support Vector Machine (SVM) on the MNIST dataset. On one hand, we explore various hyperparameters in order to know their effects on the model built with the given dataset, then set up a well performing model in term of prediction accuracy and execution time. On the other hand, we also compare these three methods on multiple aspects so as to understand more about each method's efficiency regarding the MNIST dataset and the potential scalability towards other datasets with larger size and higher dimensionality.

II. Implemented methods

1. K-Nearest Neighbors

a. *Exploring the number of neighbors parameter (value of **k**)*

First we try out different values of **k** to observe the accuracy of each value. ***Figure 1*** contains a plot representing this variation. At first observation with the training accuracy curve, it seems that the lower the value of **k** is, the better the accuracy. However, the validation accuracy do not share the same curve as the training accuracy : After reaching a maxima, the validation accuracy starts to decrease. In fact, generally for any use case, small values of **k** will cause noises that affect the prediction, whereas larger values of **k** will have lower variance but increased bias. In other word, by using its own training set to validate the model, small values of **k** with low bias will provide a high accuracy score, but when it comes to new data (in test set), it has higher chance to be an error, which causes high variance.

When **k** is increased, the training accuracy will reduce because of increasing bias, but the test accuracy will increase at the same time (decrease variance). Either way, when **k** becomes larger, since it has to consider more neighbors, its model is more complex, so the test accuracy starts to decrease after reaching a maxima.

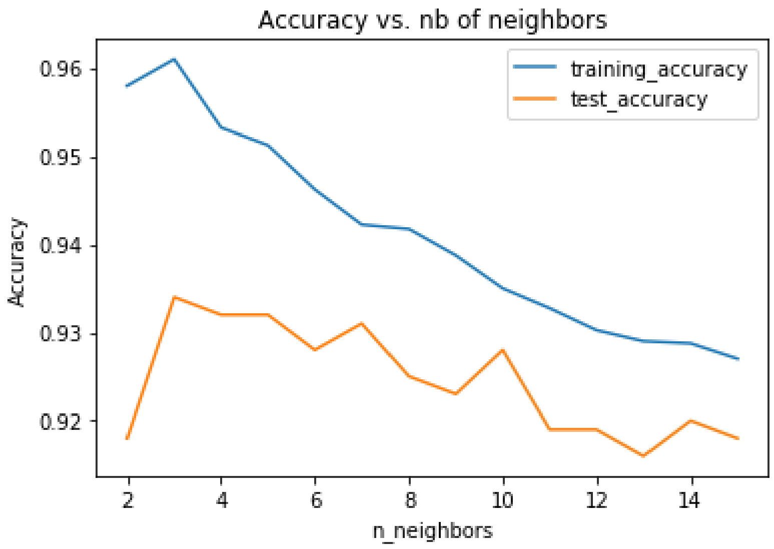

Figure 1. *Accuracy for different number of neighbors*

In order to strengthen our above observation, we move forward to the next step by performing a k-fold cross validation technique (here we use 10-fold) . What is interesting here is that, according to what ***Figure 2*** is showing, the average accuracy on each value of **k** still follows the test accuracy curve we mentioned above.

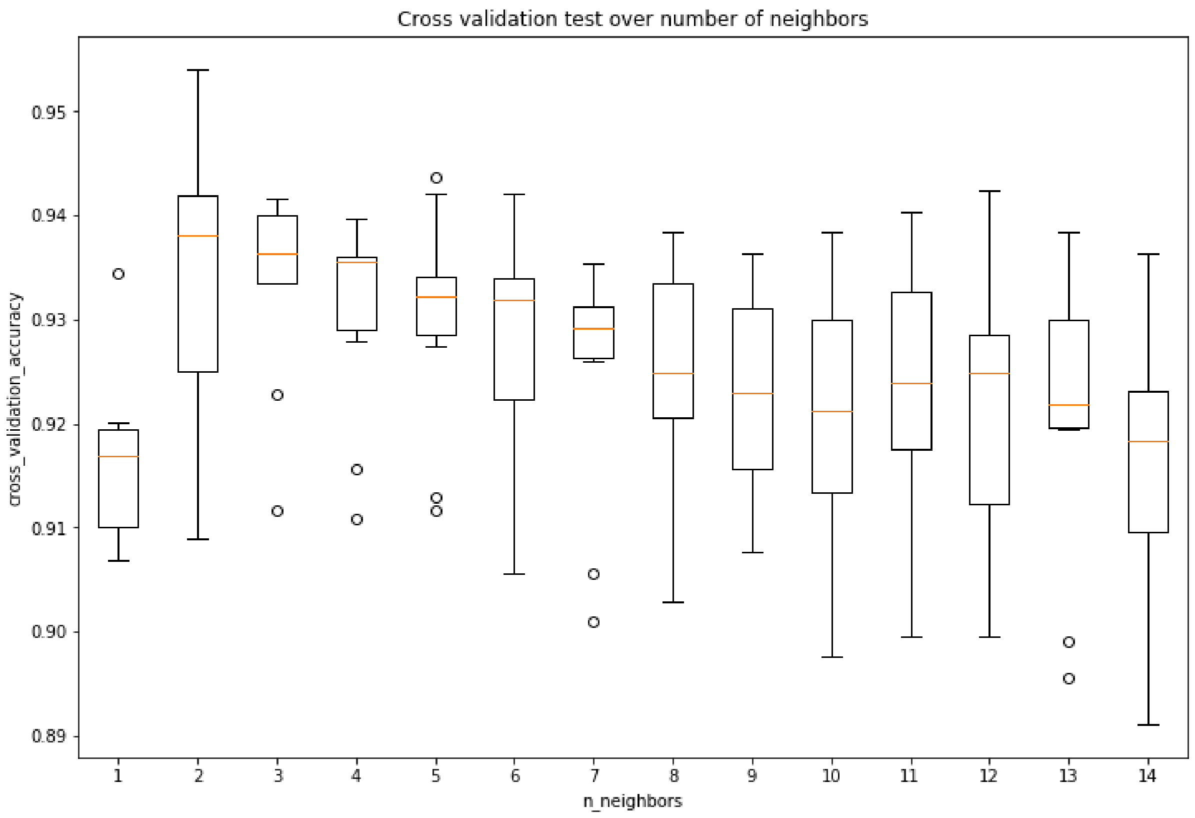

Figure 2. *10-fold cross-validation accuracy for different number of neighbors*

In short, the number of neighbors used in KNN should be not so small but also not so large. In practice, when dealing with a small dataset, **k** varies from 3 to 5 can work effectively and return an adaptive accuracy. However, for new datasets, we always need to adjust this hyperparameter in order to create a model with the best performance as possible.

b. Exploring the distance metric parameter

The distance metric is the next important hyperparameter to be taken into account. To this end, we carry out two experiments : the first one is to test the accuracy rate on models with different Minkowski distance types by varying the power parameter **p** and fixing the same **k** (here we used **k**=3), as shown in ***Figure 3***. The second one is to try out the accuracy on varying values of **k** in the 4 best-known distance metric, shown in ***Figure 4***.

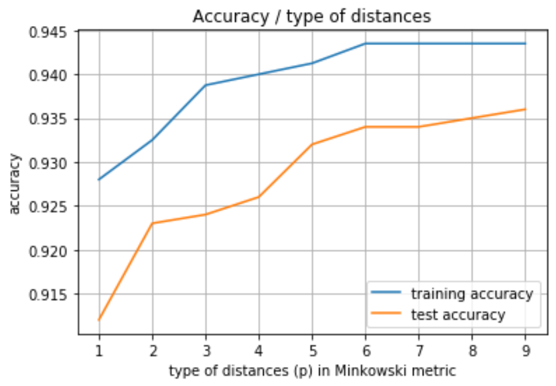

Figure 3. *Accuracy for different Minkowski metric power (**p** value) in KNN*

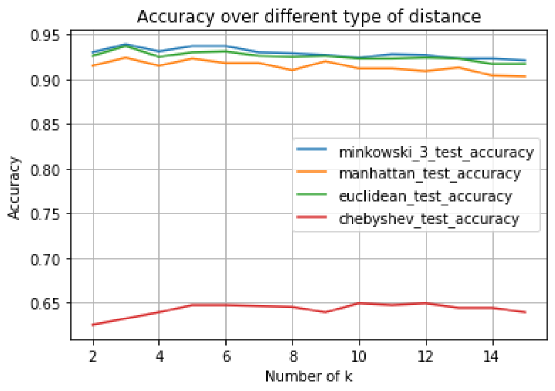

Figure 4. *Accuracy for different **k** values based on 4 different distance metrics in KNN*

Through both experiments, we can observe that the model's performance depends significantly on the used distance metric. For our use case with MNIST dataset, based on ***Figure 3***, it seems that the accuracy rate tend to increase with the increasing distance type ,in other words, the increasing power **p** of Minkowski metric. Note that increasing the power **p** will also increase the execution time of this method. On ***Figure 4***, it show that each distance metric has relatively distinct curve, where the Chebyshev metric express its far less accuracy, and Minkowski power 3 seems most accurate.

c. Training size and split rate

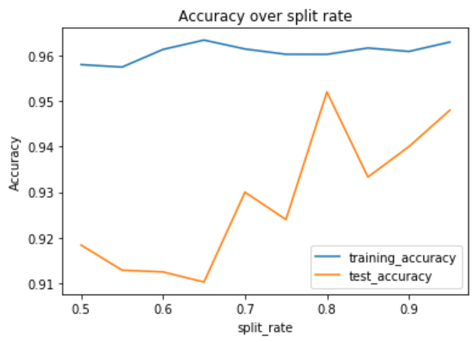

Figure 5. *Accuracy for different split rate (training size)*

Besides the two hyperparameters mentioned above, another important aspect of all machine learning problem is the data split rate to generate randomly train-test set from all the data.

By plotting an accuracy graph for different split rate ranging from 0.5 to 0.95 as shown in ***Figure 5.***, as first glance the model show a high training accuracy and an even increasing test accuracy. However, generally for any machine learning problem, when the training data portion is too high, i.e. less test data cannot represent accurately the performance of the model, it may overfit the trained model so that it fails to generalize to future dataset. On the contrary, less training data will underfit the model because the parameter estimates will have greater variance, thus decrease its performance and accuracy on test set. That is why, despite what ***Figure 5***. is showing, it is always recommended to choose a split rate from 0.7 to 0.8, where neither the bias nor the variance is too high.

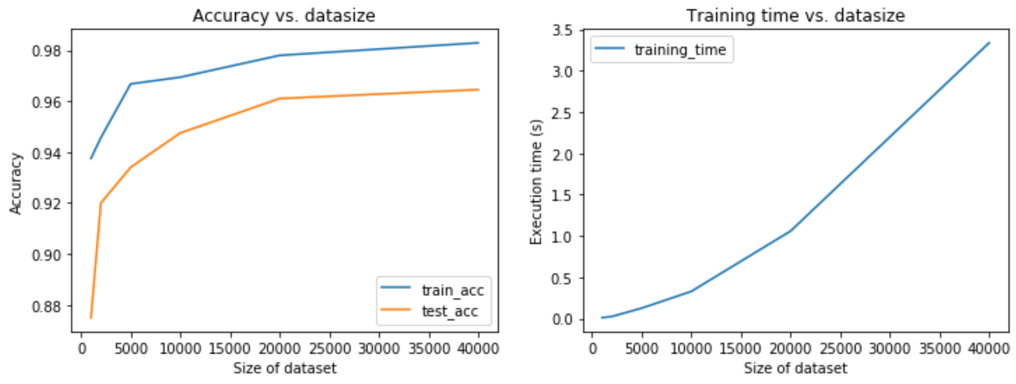

Figure 6. *Accuracy for different dataset size*

Another remark in this experiment is, as shown in ***Figure 6***, that the model is becoming more accurate when the dataset size increase, but the execution time increases as well.

d. <u>Advantages and drawbacks</u>

As the outcome of the practical work, we can conclude that KNN is easy to understand and very simple to implement with only two important hyperparameters : number of neighbors **k** and the distance fonction. Furthermore, the nature of this method requires low training time and, in case of small dataset, the classifier works effectively without complex parameter adjustments. However, the cost comes in the computational time during test phase may be high, since it has to memorize all data points, then calculate distance between the interrogating point to all other points in the dataset in order to return a prediction. For this reason, this method might not only become less effective for use in large dataset but also manifest potential problem about memory efficiency.

Further experiments also show that KNN method is rarely appropriate in case of high-dimensional dataset, when the similarity measured by the distance metric becomes less meaningful.

2. Artificial Neural Network

a. *Number of hidden layers*

Number of hidden layers is an important characteristic of Artificial Neural Network (ANN) model. For this reason we try to figure out the accuracy based on different number of layers ranging from 1 to 9. The experiment's result shows us, along with ***Figure 7***, that the fact we only vary the number of hidden layers while keeping the number of neurons within each layer at a same number does not remarkably affect the model's accuracy, whereas the execution time is highly increasing by the number of layers. In reality, the configuration for number of hidden layers and number of neurons within each layer is a complicated issue that can not be solved by simple methods. So here we decided to stick with the number of layer of 3 from now on, for the upcoming experiments.

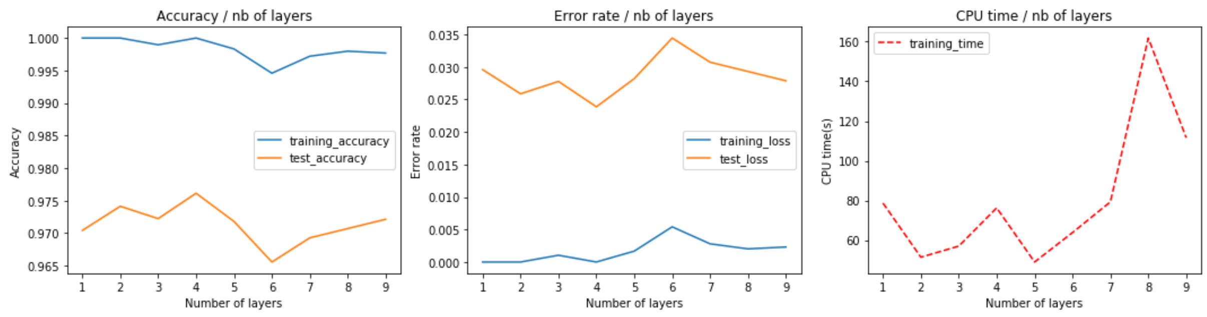

Figure 7. *Accuracy for different number of hidden layers (with 50 neurons) ranging from 1 to 9 in MLP*

b. *Optimization algorithms:*

In order to study the effect of optimization algorithms on the model's performance, we wish to observe the accuracy variation in accordance with elapsed epochs. For this end, we employ the ***partial_fit*** function so that it is trained by mini-batches, where two basic optimization algorithms ***Adam*** and ***SGD***, totally compatible to this experiment, are applied.

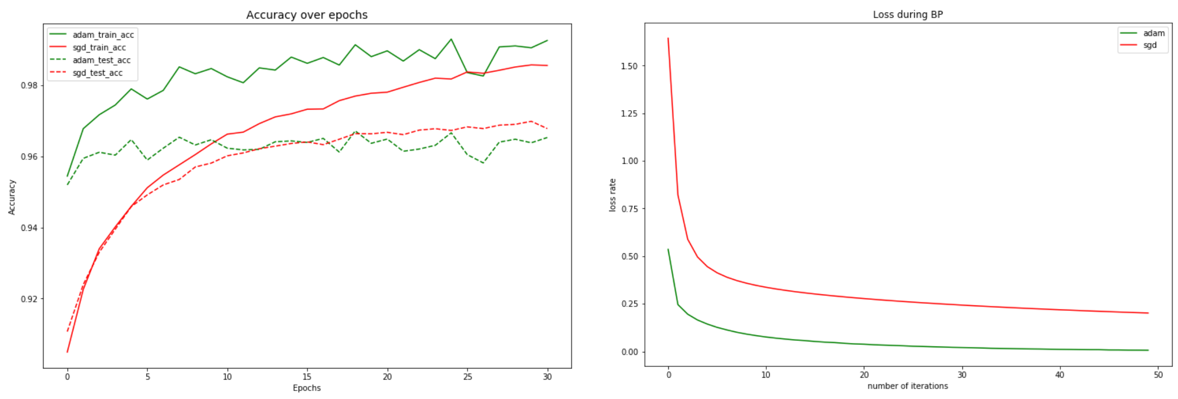

Figure 8. *Accuracy and loss variation based on two optimization algorithms Adam & SGD*

The experiments results on MNIST dataset show that, comparing to the model with ***Adam*** as solver, the model with the ***SGD*** solver has a better test accuracy over epochs (although worse on training loss), and a near-equivalent training accuracy. In this practical work dealing with a small dataset, by looking at 4 curves of ***Figure 8***, we can remark that, both ***Adam*** and ***SGD*** provide almost the same acceptable accuracy on training set and test set. Further bibliographical researches show that ***Adam*** offer huge performance gains in term of training speed, but need improvements in order to be able to generalize as well as ***SGD***. Besides, these algos are quite sensible with feature scaling, that is why the data preprocessing before training step is always important and necessary to obtain better results.

Another optimization algorithm is ***L-BFGS***, but since it is not a stochastic algorithm, it is incompatible with our experiment involving in mini-batch learning. On the other hand, ***L-BFGS*** is robust and performs better on small dataset, but it might be less effective on larger dataset or more complex models.

In general, although every method has always weaknesses, ***Adam***, ***SGD*** and their extensions are still popular optimization algorithms and they work efficiently in most cases of machine learning and deep learning problem.

c. <u>Activation functions</u>

Another hyperparameter that increases a model's performance is the nature of the activation functions. Indeed without activation function, our neural network would not be able to learn and model other complicated kinds of data. So here we try out 4 different activation functions, then, once again, use the mini-batch learning technique to observe the accuracy variation over elapsed epochs.

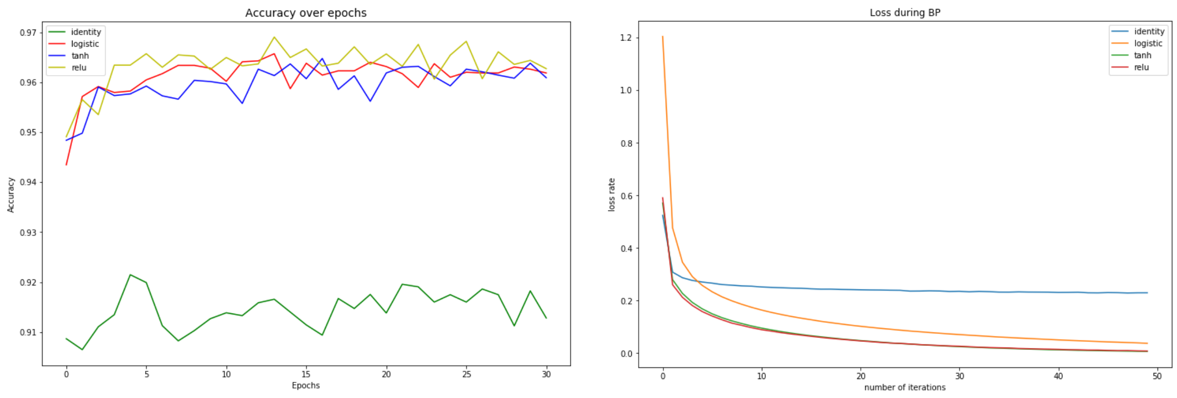

Figure 9. *Variation of model's accuracy and loss based on different activation functions used in MLP*

The experimental results, presented with ***Figure 9***, show firstly that the model with '***identity'*** function has a far less performance compared to the other three nonlinear functions. At the final epoch, overall the model with '***logistic'***, ***'tanh'*** and ***'reLU'*** function provide almost the same accuracy on test set, where the model with ***'reLU'*** attains the highest accuracy.

Since their performance results seem relatively close, we need further documentary research to find out which is the most optimal activation function to choose. In fact, the ***'sigmoid'*** and ***'tanh'*** function might encounter backpropagation error, vanishing gradient problem which lead to the performance quality degradation. Meanwhile, ‘***reLU’*** function does not have these drawbacks, on the other hand it even offer a fast model building speed. For these reasons, ***‘reLU’*** is indeed the best one out of four tested activation function, only for hidden layers, and it will be use as optimal parameter for our further experiments.

*d. Exploring parameter **alpha** of L2 Regularization*

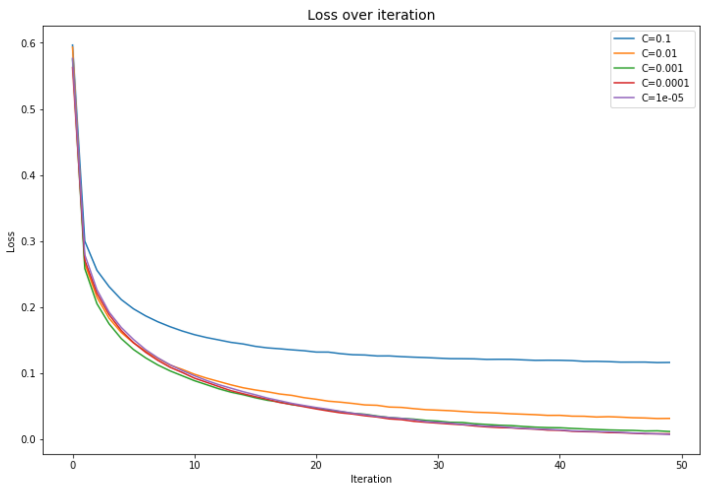

Figure 10. *Loss over iteration for each case of chosen **alpha** parameter in MLP*

While building neural network model, we might encounter underfitting or overfitting problem. Such problems can be addressed with the L2 regularization, which put penalty on the large weight of the model, i.e. constrain the size of weights during learning phase. In ***Sklearn*** neural network model, the amount of attention that the learning process should pay to the penalty is controlled by the parameter for regularization term called ***alpha***. So, increasing alpha may fix high variance, thus overcome overfitting problem, as vice versa, decreasing alpha to fix high bias, thus resolve underfitting problem. As shown in ***Figure 10.***, the smaller the ***alpha*** parameter, the less bias the model has, so its training accuracy will increase, which is equivalent to the decreasing training loss rate.

e. Advantages and drawbacks

Through this practical work, we have opportunity to work on one of the most popular machine learning technique : the Multi Layer Perceptron (MLP) - a powerful form of the Artificial Neural Networks (ANNs) which are also the base of Deep Learning models. Indeed, ANN model possesses a very high performance as it can provide high accuracy for both classification and regression problems. ANN model have the ability to learn and model non-linear and complex relationships in large size and high-dimensional datasets. For this reason, it can generalize and predict effectively on unseen data after a sufficient training time. Therefore MLP or ANN is very useful for solving stochastically many extremely complex problems.

However, the main limitation of ANN is that it cannot guarantee that the algorithm can find the global minima during training because it often converges on local minima rather than global minima. This leads to the problem of overfitting, in which the training accuracy is pretty good but the test accuracy drastically decreases. Besides, another limitation is that the number of hidden layers and hidden neurons must be set by user, hence it is not easy to build and train such optimal model. This problem remains therefore a subtle challenge which cannot be addressed with simple methods.

Also, being a very sophisticated method, it might require a large training time, as well as resources and hardware that is strong enough to handle the training phase. The input data also need to be carefully preprocessed for an efficient training, it means that data scaling will give much better results. Furthermore, the hyperparameters tuning are also proving to be very challenging. One on hand, there are lots of parameters to take into account. On the other hand, choices of optimization algorithm and activation function are not always obvious, as they depend a lot on the dataset nature.

3. Support Vector Machines

a. _Exploring different Kernel-SVM_

When using Support Vector Machines (SVM) to create a model, the kernel represents a very important parameter. In fact, since the principle of SVM is to create decision boundaries that separate different classes, this hyperparameter helps finding effectively these boundaries that are suitable to the way the data are distributed inside the dataset. In this part, we want to observe the accuracy and the training time for models using four different kernel types.

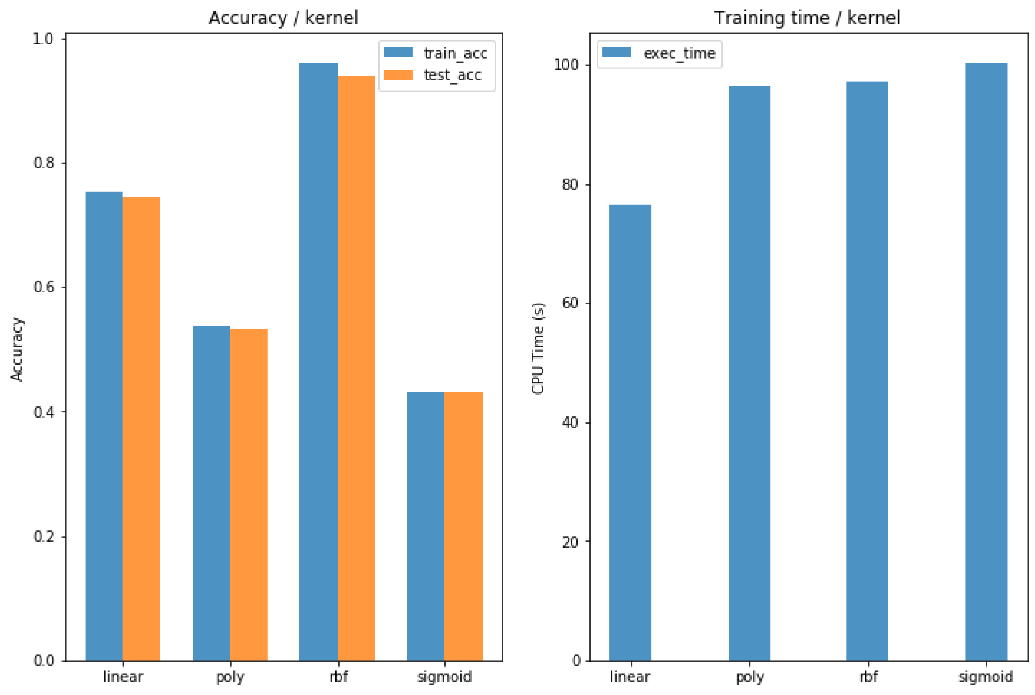

Figure 11. *Accuracy and training time for different kernel types in SVM*

As shown in ***Figure 11***, the model with '***rbf'*** kernel offers the best accuracy among 4 models. Although, like two other non-linear kernel : ***'poly'*** and ***'sigmoid'***, it has all a very long training time. The reason is that SVM with non-linear kernel has a more complex optimization problem with more variables. Hence they need a large number of iterations so as to solve this problem and look for the global minima. SVM with linear kernel, on the other hand, express its value with a faster training time, but its accuracy is far lesser than the model with '***rbf'*** kernel. This result suggests that data classes from MNIST dataset seem cannot be linearly separated, in other words, they cannot be separated geometrically with a hyperplane. However, in case of a dataset that is large, high-dimensional but quite sparse, linear kernel will become very effective since it offers much faster execution time.

b. Exploring penalty parameter C of error term

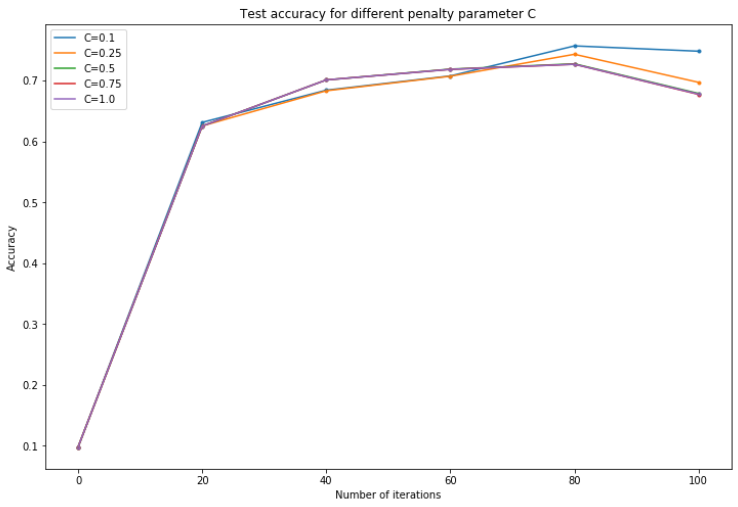

Figure 12. *Accuracy for different penalty parameter **C** of the error term in SVM*

When building a SVM model, there are two concerns : setting a larger margin and lowering misclassification rate. For this end, SVM proposes a penalty parameter ***C*** of the error term which allows to adjust margin, thus avoid misclassifying. So here we experiment on 5 different ***C*** value ranging from 0.1 to 1.

The results in ***Figure 12*** show that at the end of the experiment (here we set only after 100 iterations, but in practice, we need more iterations to better converge into the optima), the model with ***C = 0.1*** has the highest accuracy. It suggests that the larger-margin separating hyperplane, which comes from small ***C*** values, is very suitable to MNIST dataset. For additional information, large ***C*** values will cause the model to look for a smaller-margin hyperplane, i.e. a curving shape decision boundary, if it does a better job of getting all the training points classified correctly.

c. Advantages and drawbacks

Through this experiment, we firstly observe that SVM is very useful with large variety of dataset and works effectively with low-dimensional dataset , especially with the linear kernel. However, the use of non-linear kernel in this case will increase the complexity during the training phase, thus causes a much longer execution time and even memory overload. Another limitation of SVM is that there is a need of a data preprocessing step as well as the precise hyperparameter tuning to perform a better accuracy.

Overall, SVM is very versatile because it provides different kernel functions so that it can be more suitable to the chosen dataset and improve the model's performance quality. It also allows to specify custom kernels when needed. Furthermore, the nature of SVM is to use a subset of training points in the decision function - called support vectors, so it is memory efficient.

Nevertheless, in case of dataset with uncommon distribution, by example it is not linearly separable, the choice of kernel function and regularization term is crucial, also the cost for using non-linear kernel has to be considered.

III. Comparison of studied methods

Performance

In order to compare ML classifiers' performance, we can refer to: *(i)* performance of a classifier itself, i.e. the execution speed of the model builder or training time and *(ii)* performance of the predicate step, i.e. the the execution speed of the model when predicting

From our experiments on MNIST-784 dataset, we realize that SVMs and ANNs are much slower than kNNs in term of training time. There are some reasons for this : SVM training requires solving the associated Lagrangian dual problem, which is a quadratic optimization problem with very large number of variables. ANN requires also a high training time, especially for complex and strong models because of the large amount of layers and neurons within these layers. This is why SVM and ANN models need a large number of iterations so as to have better convergence to the optimum. The more number of iterations is, the longer training time is.

However, the predicting performance of a SVM is substantially better compared to ANN or KNN. Indeed, for a 3-layer MLP, the prediction requires a forward-propagation with successive multiplications, whereas for a SVM classifier just involves determining on which side of the decision boundary a given point lies.

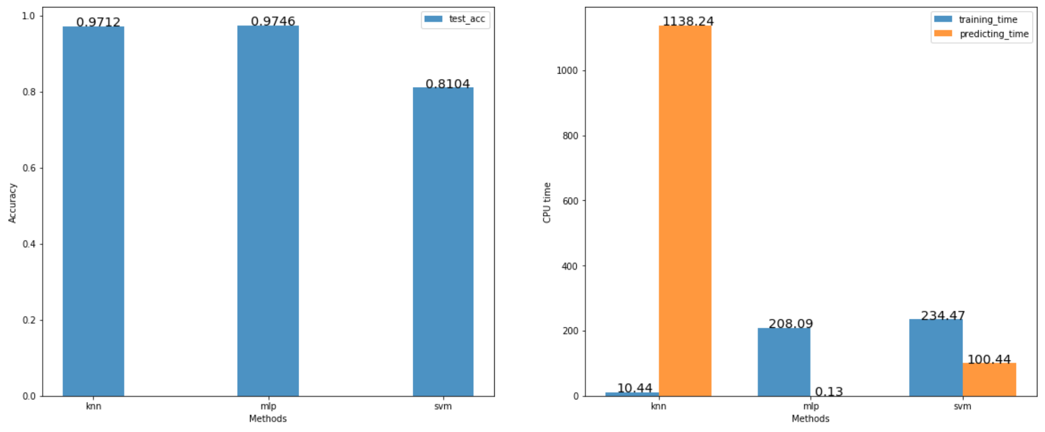

Figure 13. *Comparison of performances & accuracies between studied methods*

Accuracy & Scalability

Besides, another important metric is prediction accuracies which is used to compare between classifiers. Thus, from the practical works, we realize that all the accuracies of these three methods are pretty good. To have a general look : KNN performs well for small datasets (<100000 samples, non-contextual) because of its simple use with only two important properties (number of neighbors & distances). SVM works efficiently for small and medium dataset with similar features. ANN has evolved over time with complex models and it is powerful for large-scale dataset. However, SVM and ANN are very sensible to the hyperparameters and they also require that the data are scaled during preprocessing step before training phase in order to obtain a better accuracy.

In summary, all aforementioned methods (i.e. KNN, ANN or MLP, SVM) are three popular strategies for supervised machine learning and classification task. However, it is not often clear which method is better for a specific project. Therefore, when we work on a project with a new dataset, it is often preferable to start by simple models such as a k-nearest neighbors classifier or a linear model, and then we try to figure out where we should go next. When we can understand better the dataset based on mentioned simple methods along with exploratory data analysis, we can pass to a better algorithm which is able to build complex models such as SVM or Deep Neural Networks (MLP is its simplest version).

IV. Conclusion

Through this practical work, we have discovered three different machine learning method with three totally different nature, more precisely :

- K-Nearest Neighbours method which is popular for being very simple and intuitive to understand and to implement with small dataset.
- Artificial Neural Network as a very sophisticated method, capable of providing most of the time a high performance model dealing with larger and high-dimensional dataset, and as the base of Deep Learning field.
- Support Vector Machines which represents the machine learning's geometrical approach and express its complexity facing complex dataset.

Here, not only we experimented three different techniques to see which provides the most optimal model for the MNIST dataset in particular, but also observed each technique's eventual behaviour facing different dataset types. That is why we remarked that exploration data analysis consists of a crucial step before proceeding to any machine learning technique application so as to obtain a good result in term of model's performance.

We also realize the importance of carefully configuring differents classifier properties. Indeed, each machine learning classifier has its own hyperparameters that have huge impact on the model's accuracy. An arisen challenge here is not only to obtain a model that is suitable to some specific use cases, but delicately, as most generic as possible for new data.