Machine Learning Project Milestone 2 Report
347304_314355_345583_project
Nikolay Mikhalev, Erik Georges, Gafsi Amene

Introduction:
In the second part of this project, our aim was to implement a few more sophisticated machine learning models More specifically, two deep learning algorithms: namely MLP and CNN. These two models, even though more computationally heavy, tend to outperform the linear models that we've implemented in MS1 thanks to the hidden layers and the multiple nodes that could be trained well enough as long as the dataset is large enough and is coherent. In addition to these two models, we implemented PCA, a method of data dimension reduction. In this report we present the different techniques used to augment the performance of these models and the results obtained.

PCA:
Since this technique of dimension reduction doesn't really function with CNNs and MLPs, we started by implementing PCA and testing it on the previous models. This method helps drastically reducing the time needed by reducing the data dimensions. The results presented in the next chart show that even though in our case, PCA did not particularly help improve the accuracy of our models, it still did help reduce the training time. The times presented are the times needed for running cross validation and the dimensions left by PCA are 300 (summing up to 0.9 explained variance). If we take into account that we are working with 32x32 = 1024 dimensions, a division of the data by a factor larger than three, while guarding the same accuracy could be perceived as a win. For PCA to show even better results, we suppose that the dataset should possess more dimensions to start with. For example, reducing 15'000 dimensions into around 500, PCA could

	Accurcy Without PCA			Accuracy With PCA (300 dim left)		
	Training	Testing	Time [s]	Training	Testing	Time [s]
K means	65.59%	62.40%	5.49	65.34%	62.17%	2.88
Logistic regression and cross validation	100%	84.43%	378.92	100%	86.28%	150.717
SVM with linear kernel and cross validation	97.49%	90.08%	245.21	96.61%	90.20%	75.66
SVM with rbf keernel and cross validation	100%	92.04%	3264.86	100%	93.20%	1381.4
SVM with polynomial kernel and cross validation	100%	92.73%	8731.845	100%	93.08%	2524.384

really help, keeping the variance in data high and largely reducing the computation cost.
As we can see in the table presented here, not only have we gained a small accuracy boost, but the time reduction brought by PCA is really felt, when the computations get heavy. For example, using cross validation on SVM with a polynomial kernel, we had 300 combinations of possible hyper parameters to test, which, without PCA took around 2h20min to be done, with PCA this time has been reduced to around 40min and even yielded a slight augmentation in the testing accuracy.

MLP :
The MLP (Multi-Layer Perceptron) model is an artificial neural network used for classification tasks. It comprises multiple layers of interconnected neurons that apply non-linear activation functions to input data. This enables the model to capture complex patterns and relationships in the data.
For our MLP implementation, we designed a network with three hidden layers. The sizes of these hidden layers were set to 512, 256, and 128 neurons, respectively. We applied then rectified linear unit (ReLU) activation function to each layer, which helps introduce non-linearity into the model and allows it to learn complex patterns in the data.
By employing our custom-coded cross-validation algorithm, we successfully optimized our model to find the best parameters. Notably, after training, we found that a learning rate of 0.01 and a maximum iteration of 100 significantly enhanced accuracy, resulting in an approximate 91%

accuracy on the test set. These findings underscore the effectiveness of our cross-validation algorithm in identifying optimal hyperparameters for maximizing model performance.

CNN:

A convolutional neural network is similar to a MLP but takes advantage of the fact that the input data is made of 2D images which allow the use of convolution to decrease the size of the input and thus decrease the number of weights required by the network which can become very big in a MLP.

Our CNN implementation had two convolutional layers that use max-pooling and a kernel of size 5, and three fully connected layers with a ReLU activation function applied between each of them. We chose convolutional layers with 10 and 20 channels, and fully connected layers of sizes 30, 40 and 20 (the output size). The hyperparameters (learning rate and maximum number of iterations) were optimized using the cross-validation method, the results we obtained were the following:

max.number of iterations	learning rate	F1-score
100	0.01	0,917
100	0.001	0,086
100	0.0001	0,014
100	0.00001	0,009
500	0.01	0,920
500	0.001	0,895
500	0.0001	0,031
500	0.00001	0,009
1000	0.01	0,922
1000	0.001	0,908
1000	0.0001	0,321
1000	0.00001	0,014

The hyperparameters that worked best were a learning rate of 0.01 and 1000 max. iterations, which yielded an accuracy of 94.2% on the test set. We notice that there is a big drop in accuracy when using a learning rate below 0,001 and that changing the number of max. iterations has a rather minor impact on the performance of the network (but significantly increases the training time).

Our results show that the CNN performs better than the MLP while using one less fully connected layer, as well as layers of smaller size, which shows that convolution is an effective tool when we have 2D images as input data.

Cross Validation with fold:

A very useful and technically simple tool to find the best hyper parameters for our models. If it doesn't really play any role with k-means and only small one with logistic regression, helping check that we're using an appropriate learning rate, this tool really gets handy with the SVM, where there are multiple parameters to test. One important part of cross validation was also using folds: instead of taking (for example) the first fourth of training data as validation data and the rest as training, we would iterate and in this case do 4 tests, then taking the average of the four and assigning the params in question this very score. If we didn't do this, then the information would be rather random and imprecise because of the data shuffling, we would get drastically different results each time.

Training:

Training is crucial for neural networks to learn and enhance their performance. In our case, we have developed a specialized training algorithm implemented in the Trainer class to effectively train our models. This algorithm follows a simple process where it initializes essential components like the criterion and optimizer. It then proceeds to train the model over a specified number of epochs by iterating through the training data in batches, performing forward passes, computing losses, running backward passes for gradient calculation, updating weights, and zeroing out gradients. During prediction, the algorithm sets the model to evaluation mode, iterates through the test data, and makes predictions. The fit method prepares the data, trains the model, and returns predicted labels for training data, while the predict method prepares the test data and returns predicted labels as a numpy array. Overall, our algorithm has achieved excellent training results for our neural networks.