Une image contenant texte

Description générée automatiquement

Paul Batut

2023

Machine learning project

Summary:

1. Introduction page 2
2. Workflow of the machine learning process page 2
   1. Train/test split page 3
   2. Data Examination page 3
   3. Data Preprocessing page 3
   4. Model building page 4
   5. Model training page 5
   6. Performance tuning page 5
   7. Evaluation page 6
3. Comparison between MLP and SVM page 7
4. Conclusion page 7
5. **Introduction**

In this machine learning task, we were required to build a model that was based on a dataset that was given to us in order to classify different wireless transmitters. In this report, I will explain the different steps I took to create my model, as well as explaining the different graphs that I got.

For this project, we were provided three different files: x\_train.csv, y\_train.csv as well as x\_test.csv. Those files were apart of the dataset that we exploited during this whole project, and each had a different role.

* X\_train.csv is the file that held every information for each variable. Indeed, as stated before, the goal of this project is to classify different types of wireless transmitters, which have in total 12 attributes.
* Y\_train.csv is the file which assigns an output, or target, to an id. It is thanks to these data that the model we will create will be able to categorize the tested data.
* X\_test is the final file and it looks quite similar to X\_train.csv. This can be explained as the data are used to try the model and make sure it works properly.\*

1. **Workflow**

Once we retrieved these data, we can begin the model building workflow.

*Une image contenant diagramme

Description générée automatiquement*

1. **Train/test split**

As seen in the previous image, the first step in this the process is the test and train split. We already x\_train and y\_train on one side and x test on the other, so this is it for the test/train split.

1. **Data examination**

data examination, which consists of opening the x\_train csv file and getting different information about the dataset, such as its size (meaning the number of samples as well as the number of features), the different head of columns which are the dataset’s features names, and in our case, we displayed data from the first 5 rows as well as from the last 5 rows. This part is essential in order to understand our data better , and not getting all the necessary information can lead to misunderstandings.

1. **Data preprocessing**

Next is the data preprocessing. The first part was us reading the data, and this part is us modifying these data to keep the necessary information.

As some data will be modified, we create some variables whose purpose will only be retrieve the csv file, which will be copied by other data so that the original data stays intact.

Once everything is retrieved, we check the second column of y\_train.csv and retrieve all the possible targets (which were stated to be from 1 to 8) thanks to the column name. Using this value, we used get\_dummies (which converts the data into binary variables with as many columns as there are classes, and the class one ID might have becomes 1, the rest is 0) to get a good classifying approach (as we are doing a classification model). Then the next thing we did was to remove several classification values. Indeed, I believed that the more data we had, the more accurate our model would be therefore we removed as little data as possible. Tosc, Tmix and m\_power were all supposed to be removed, according to the subject, and Unnamed: 0 was supposed to be a variable called cfo\_meas therefore we decided to remove this target (as it was also specified in an email that it was not that important).

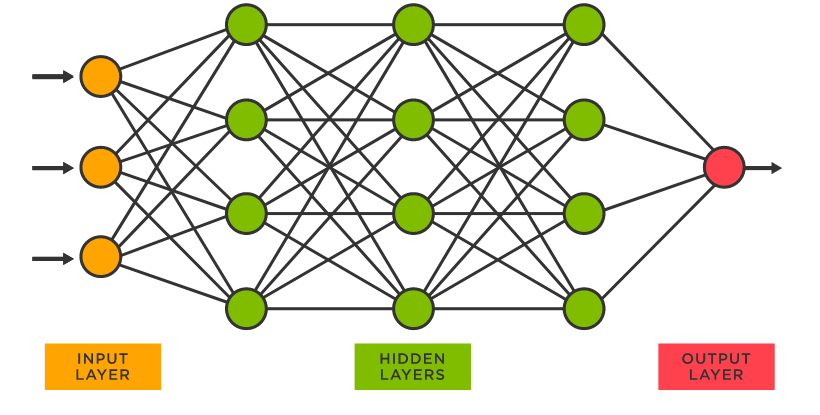
Finally we classify the values that are present in the x\_train.csv file into different classes so that those data can be exploitable.

1. **Model building**

Now comes the part of the model building. The model is the core of the whole project and is the artificial intelligence that will process the data and give all the predictions of the targets for one id.

However it is also the most complex part of the code: there are several parameters (called hyperparameters) that can be tweaked to get increasingly better results.

The neural network that represents our model can look like this:

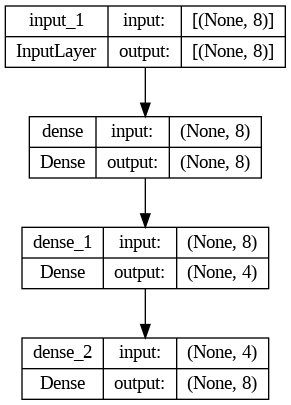


The input layer of a neural network is where the data comes in, and it is then transmitted to the hidden layers, which can be as many as needed. Finally, when the signal has done the whole path it was required to do, it comes out on the output layer that may have one or more neurons.

In the case of our neural network, we decided to have a sequential model as it seemed the most fitting. We have had an input layer that had 8 neurons, as we had 8 classification values, an output layer that had 8 neurons, as there were 8 targets, and we tried several combinations with the hidden layers.

As it was stated on the subject, we tried to keep the neural network as simple as possible, so we decided not to go above three hidden layers. Some more explanations will be given at the hyperparameter tuning phase.

We found out in the end that having two hidden layers where one hidden layer has 8 neurons and the other has 4 gives us the best results, so we decided to keep this configuration. We also found out that a ReLu (Rectified Linear) activation method was far more effective than sigmoid (which gives out a signal between 0 and 1) for the hidden layer, however sigmoid is necessary for the output layer as it gives us the weight of one output, helping the model to choose the best possible option. Therefore, our model looks like this:



1. **Model training**

The next part is the model training part, where we first split our data with a test size that is of 0.2, meaning that 80 percent of the data from the dataset will be fed to the model and 20% will be used to check the efficiency of the model. With this split, we get the x values for training and testing, and the same goes the y values.

We then defined our optimizer to use the Stochastic Gradient Descent (SGD) method with a learning rate of 0.01, as we saw in previous labs that it was the optimal learning rate.

We chose to use the binary crossentropy loss function with the metrics being set to accuracy.

Finally, we fed the model with 300 epochs, and with a batch size of 300.

1. **Performance tuning**

As stated earlier, there are several parameters that can be tweaked to get better results. These parameters in our case are the number of neurons in the neural network, the activation functions, the learning rate, the number of epochs as well as the batch size.

We did several tests trying to alter the number of epochs. We found out that at a certain point, too many epochs will not change anything and that we should keep a relatively low epoch number. We ended up keeping 300 epochs, which still takes around 20 minutes with a batch size of 16.

Our neural network ended up having two layers, one with 8 layers, one with 4 layers. But to keep this result, we have had quite a lot of tests, all of which can be seen on the google collab.

1. Evaluation

We have these graphs that show the loss function as well as the accuracy:

Une image contenant graphique

Description générée automatiquement

*Loss function graph*

*Une image contenant graphique

Description générée automatiquement*

*Accuracy graph*

As we can see, there are two increases in accuracy that coincide with the loss drops.

1. **Comparison**

We tried using SVM (Support Vector Machines) to create a model compared to MLP (Multi Layer Perception) that we previously used, by using similar parameters.

We ended up checking the score which was really good with 0.9997, therefore it seems like SVM is a better solution

1. **Conclusion**

In the end, we managed to build a model that looks like it is quite efficient, with an accuracy that goes above 90%. As I’ve seen on the Kaggle leaderboard, there are still ways to improve this model as some people have gotten some scores that were above 99%, but we’re quite proud of our scores, as we’ve been stuck with an accuracy that was stuck at 37% for quite a while, before finding out that using relu instead of sigmoid for the activation functions was much more effective.

This course and this were really interesting as it really confronted us to what we learned earlier in this semester with a project that had a real life application, and made us understand better machine learning where we sometimes had issues understanding some of the code. Doing it all ourselves and doing research to get the best possible parameters while trying to understand all it did was really enriching.