COMP6208 - Advanced Machine Learning
Higgs boson machine-learning challenge

Citraro Leonardo	Perodou Arthur	Iyengar Akhila	Roullier Baudouin
lc4g14@soton.ac.uk	alp2e14@soton.ac.uk	ai4g14@soton.ac.uk	br1u13@soton.ac.uk
ID 27393224	ID 27266303	ID 27014487	ID 27414701

May 6, 2015

1 Introduction

For our machine learning project we decided to apply our knowledge on the Higgs Boson challenge. The contest has been provided by the CERN (European Council for Nuclear Research) in order to establish a collaboration between high energy physicists and data scientists. The objective of the collaboration is optimize the analysis and discover of the Higgs boson particle. The ATLAS detector simulator provides simulated data events based on proton-proton collision imitating real statistical properties. Each simulated event could belong to two different classes: signal or background. The formal objective of the competition is developing a classifier which provides the best approximation of the median significance (AMS).

1.1 Aims and Objectives

In this project we aim to increase our knowledges in machine learning and to achieve a good result/rank in this challenge. The main objective is to implement several different classifiers (Support Vector Machine, Neural Network), improve and compare them in order to state virtues and vices of each algorithm for this specific problem.

2 Data Description

The dataset for this challenge comprises of a set of 250'000 events for training and 550'000 for test. The training set contains an event ID, a label and weights associated to each sample. This dataset consists of a set of simulated events produced by CERN using the ATLAS full detector simulator. Simulation is done basically in two parts. In the first section, random proton-proton collisions are simulated and in the second part these proton particles are tracked using a virtual detector. This process results in simulated events that mimic all the necessary properties of the real events. The data consists of signal and background events, where signal consists of Higgs Bosons and background is randomly generated using 3 processes involving of the decay of Z boson (similar to decay of Higgs), events with the pair of top quarks involving decay of lepton and hadronic tau and finally the decay of W boson (electron/muon+hadronic tau). Special care was taken when simulating these events to make sure that the number of simulated signal and background events proportion does not correspond to the prior probabilities. This might frequently happen because the number of background events is more than the number of signal events. In order to make the simulation process easier, each event has a non-negative weight associated with it. These weights are proportional to the conditional density over instrumental density of the simulator. However, the objective function relies on unnormalized sum of weights. In order to make the setup invariant to the number of signal and background events, the sum across each set are kept constant. In our case, weights correspond to the quantity of real data. These are then normalized (N_s, N_b) for integrated luminosity in such a way that that sum of weights falling in a particular region is an unbiased estimator of the expected number of events in the same region. There are many features, which characterise these events, out of which the most relevant 30 features are considered for the challenge.. Almost all the variables in the data are floating numbers expect the one that is specified by **PRI_jet_num** which are integers.

3 Evaluation

The goal of this project is to classify the events into two different classes, called signal and background. Because the events are not all equally significant, the efficiency of a classifier is measured with the approximate median significance, or AMS for short, which takes into account the weights associated to each sample.
From these weights, we calculate the unnormalised true positive and false positive rates, respectively s and b, considering the prediction and the truth:

$$s = \sum_{i=1}^{N} w_i \mathbf{1}\{y_i = s\}\mathbf{1}\{\hat{y}_i = s\} \qquad b = \sum_{i=1}^{N} w_i \mathbf{1}\{y_i = b\}\mathbf{1}\{\hat{y}_i = s\}$$

Where w_i is the weight, $\hat{y}_i$ is the prediction, and y_i is the true label, for the i^{th} event, and the indicator function $\mathbf{1}\{A\}$ is 1 if its argument A is true and 0 otherwise. So s counts every correct signal prediction, and b counts the events we predicted as signal but are in fact background.
From these two variables, it is possible to calculate the AMS with the following formula:

$$\text{AMS} = \sqrt{2\left((b+s+10)\log\left(\frac{s}{b+10}\right)-s\right)} \tag{1}$$

The AMS for this competition lies between 0 and 4. The winner of the Higgs Boson Challenge reached an AMS of 3.8 on the test set.

4 Support Vector Machine (SVM)

4.1 Implementation

This section aims to describe and explain the choices made on the final Support Vector Machine classification framework.

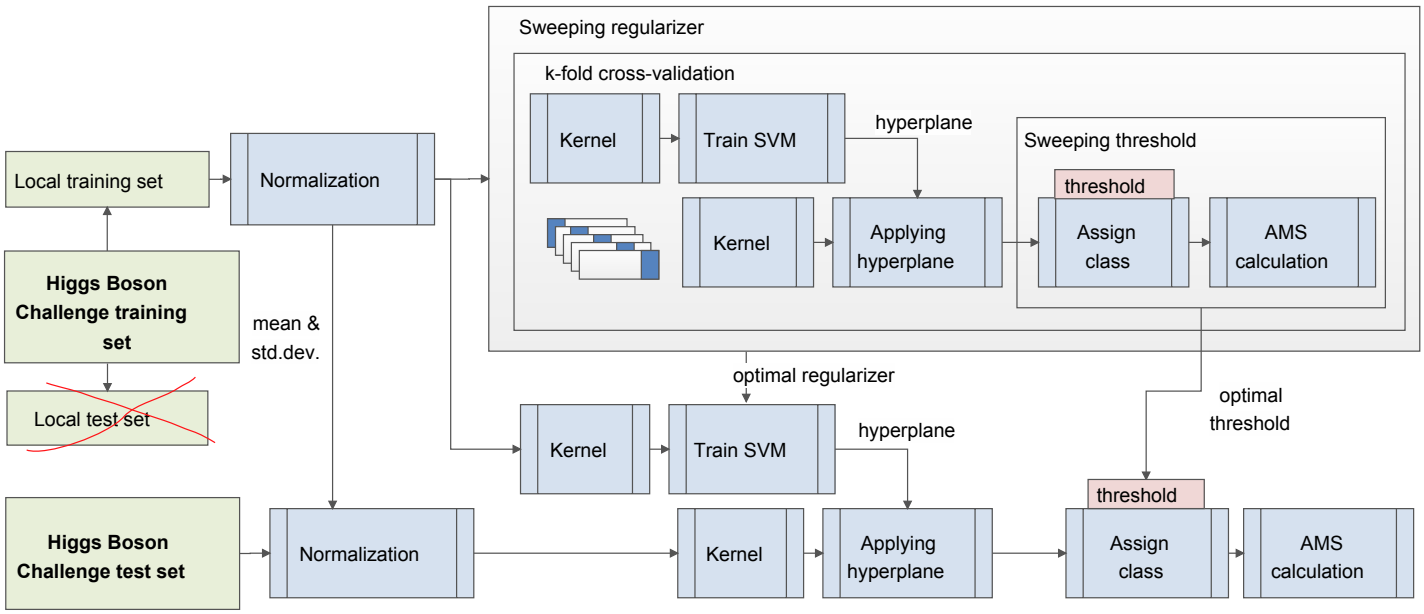


Figure 1: SVM framework

At the beginning of this project the official Higgs Boson Challenge test dataset was not provided to the public, however, after a month the dataset has been released, this explain the red cross on the SVM diagram. The Higgs Boson Challenge training set provided by the Kaggle website [1] has been split in one local training set and one local test set at the beginning of the project but, after the release of the official test set, we just used the same framework substituting the local test set with the official one. We have not re-trained the SVM with the entire training set because it is really time consuming and also due to the numerous number of runs to run for plotting the graphs in figure 3. The local training set has been used for training and validation whereas the test has been used for final testing purposes. In each subset we modified the weights in order to have the same sum of signal-weights s and background-weights b according to the original dataset. The training set has been normalized (zero mean unit standard deviation) in order to optimize the training process and speed of convergence. A k-folds cross-validation (CV) allows us to calculate a more precise AMS compared to a simple holdout CV. Each fold pass through the chosen kernel and through the SVM, the hyperplane calculated is then applied to the validation set beforehand mapped by the same kernel. This resulting continuous value vector is then compared to a threshold which defines the class of belonging of each event (signal or background). The "sweeping threshold" process finds the best threshold giving the maximum AMS where the optimal threshold is extracted by averag-

ing the k AMS curves. The training process's output is the optimal regularization coefficient (λ) and the optimal threshold.

On the test path, the entire training set is trained considering the optimal regularizer λ. The test set has been normalized using the same mean and standard deviation used for the training set and mapped using the new calculated hyperplane. The class assignation uses the optimal threshold calculated beforehand and the final AMS is calculated consequential.

VLFeat [4] provides a very powerful C-coded toolbox for MATLAB containing the Support Vector Machine and several Homogeneous kernel maps. The homogeneous kernel map [5] performs an approximation of non-linear mappings based on a spectral analysis, therefore, it is faster and is more suitable for large scale datasets. The kernels provided are the chi-squared: $K(x,y) = 2\frac{xy}{x+y}$, the intersection: $K(x,y) = min\{x,y\}$ and the Jensen-Shannon: $K(x,y) = \frac{x}{2}log_2(\frac{x+y}{x}) + \frac{y}{2}log_2(\frac{x+y}{y})$. Due to the size of the dataset, a pure non-linear kernel like the Radial Basis Function cannot be implemented easily in MATLAB. We attempted to map the dataset using a non-linear transformation but the physical memory required is much more than what we dispose.

4.1.1 Compute optimal threshold and AMS

To compute the optimal threshold and to calculate the cross-validation AMS we simply average the k "AMS vs. threshold" curves. Figure 2 shows the k curves, the mean and the standard deviation. The resulting AMS is the maximum value of the averaged curve and the threshold defines this quantity.

4.1.2 Choosing k of k-folds cross-validation

In order to chose a good number of folds for our cross-validation, we ran ten times our training algorithm and extracted the maximum AMS and threshold form the averaged curve for different values k. The maximum AMS for k=10, 5 and 2 is $\mathbf{2.017 \pm 0.0043}$, $\mathbf{2.021 \pm 0.0046}$ and $\mathbf{2.012 \pm 0.0064}$ respectively. The optimal threshold for k=10, 5 and 2 is $\mathbf{0.301 \pm 0.0357}$, $\mathbf{0.305 \pm 0.0306}$ and $\mathbf{0.287 \pm 0.0334}$ respectively. The difference from the maximum AMSs and the difference from the thresholds considering k=10 and k=5 is minimal whereas for k=2 we can see a slightly increased variance. In order to chose the parameter k, the time required for training is also a parameter that matters, more folds means more time. Therefore, for the SVM framework we chose k=4 which is a good compromise between variance and time required.

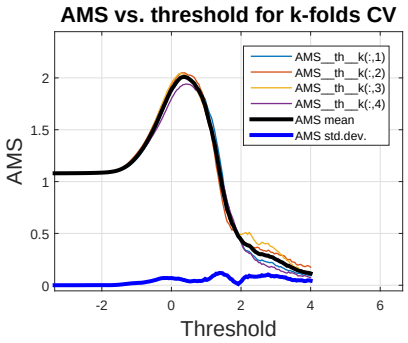


Figure 2: Example of k AMS vs. threshold curves, mean value (black) and standard deviation (blue)

4.2 Experimental Results

The SVM has been setup using the fastest solver and loss: Stochastic Gradient Descent (SGD), Loss function HINGE and the default Slack of the margin criterion epsilon equal to 1e-3. The maximum number of iteration has been set to 1e8 in order to ensure the convergence. The kernel has been set-up as follow: Rectangular window, default Homogeneity degree gamma equal to 1, and the order has been set to different values. The kernel used are the chi-squared (Kchi2), intersection (Kinters), and Jensen-Shannon (Kjs).
Figure 3 shows different results obtained by sweeping the regularizer parameter λ for different kernel types. The maximum result obtained without a kernel is **2.05 AMS**. The 5th order kernel Kinters scored the highest result of **2.66 AMS** followed by the 5th order Kchi2 at **2.64 AMS**. As we can see, the use of the kernel clearly improve the final results, however, the order does not seem to have a considerable effect. The fifth graph on the right shows the same curve AMS versus the regularizer but considering less features. We used the Principal Component Analysis (PCA) to perform the reduction of the dimensionality and we performed the same test in order to state whether it is possible to discard some features. The results using 25, 28 and also 30 features is considerably less than the AMS seen before. This result helps us understand that the features are likely to be useful but in order to consolidate this result we have to implement other type of feature selection as the Linear Discriminant Analysis (LDA) or Embedded Methods.

SVM type/results	Validation AMS	Test AMS
SVM 5th Kinters	2.66	2.44

Table 1: Best result on validation and test sets considering the SVM with kernel

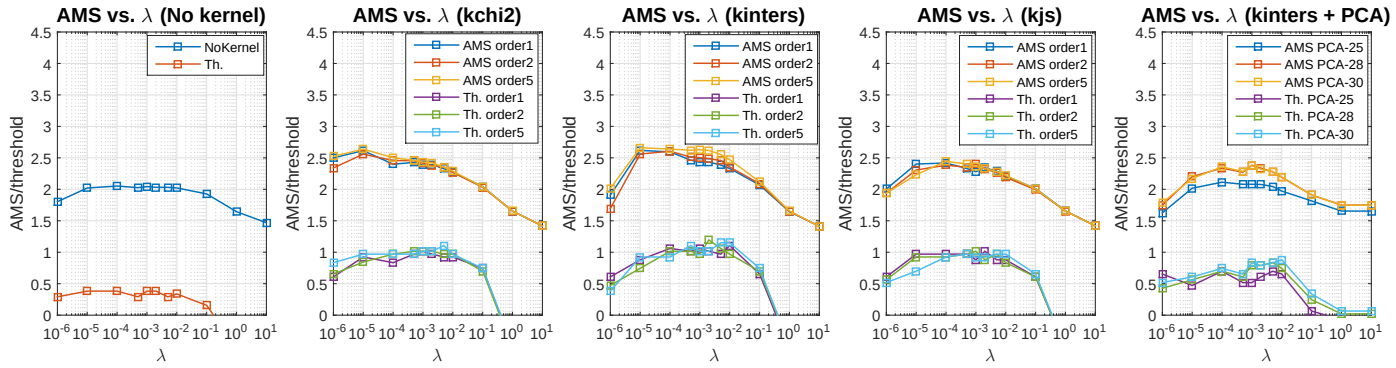


Figure 3: Maximum AMS against λ and the associated threshold (Th.). From left to right: (1) linear SVM only (2) SVM + kernel Kchi2 order 1, 2, and 5 (3) SVM + kernel Kinters order 1, 2, and 5 (4) SVM + kernel Kjs order 1, 2 and 5 (5) SVM + Kernel Kinters order 1 considering projection on 30, 28 and 25 principal components.

5 Neural Network (NN)

In this section, we deal with Neural Networks to solve the Higgs' Boson binary classification problem. In this project we used and modified the Deep Learning Toolbox [7] proposed by *Rasmus Berg Palm*. This toolbox is very well coded and it is faster than the basic *Matlab* Neural Network Toolbox.

5.1 Single Layer Neural Network

5.1.1 Implementation

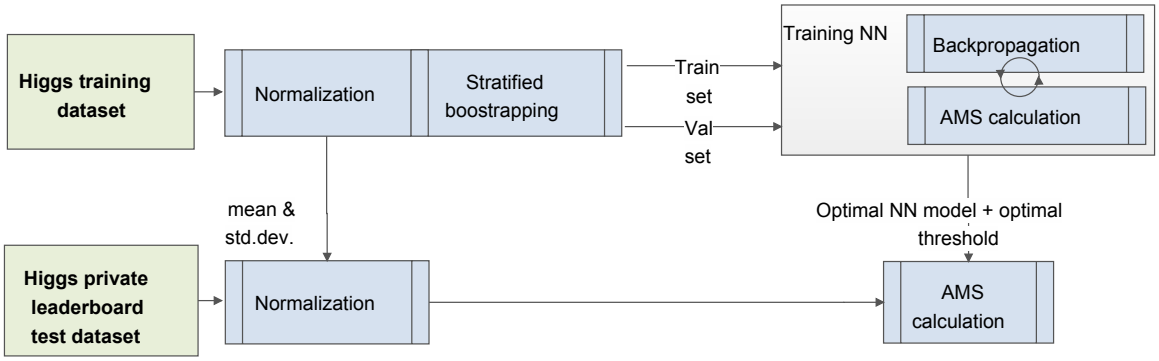


Figure 4: Single layer NN framework

Figure 4 shows the framework of our single layer Neural Network. After the normalization of the training set, the data are split into a training and a validation sets using the Stratified bootstrapping function. Successively, the network is trained using the training set only. At each epoch, the AMS is calculated on the training and validation set by sweeping the output threshold for a large panel of values. Figure 5 shows an example of AMS curves extracted during one training process. The weights of the NN are stored every time the AMS of the validation set is greater than the previous result. This approach is somehow similar to the early-stopping method and it guarantees to have the best weights at the end of training process even if the number of epochs are greater than what it is actually necessary. In the basic Neural network the sigmoid function is used for hidden and output neurons. We used a mini-batch size of 150 samples for the mini-batch gradient descent algorithm, learning rate of 1.5 and 50 epochs

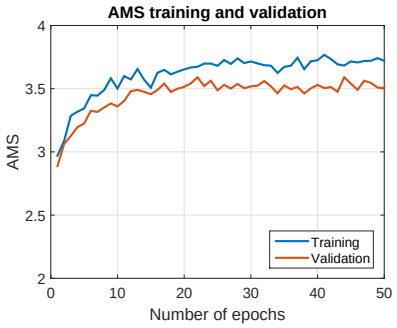


Figure 5: Example of AMS curves for training and validation set from the NN training process

Normalization: In the normalization block we applied the classic zero mean unity standard-deviation for all the features.

The mean and std. are then stored and used on the test set.

Stratified Bootstrapped k subsets: At the beginning of this project we adopted the classic k-folds cross-validation as validation process which consists in dividing the training set in k-folds without paying attention to the distribution of the weights in each subset. The training set is composed of weights for signal and background events. The weights associated to the signals are only of three kind (0.0027, 0.0186 and 0.0015) whereas the background weights are numerous and evenly distributed. The stratification approach evenly subdivides the three different weights in each sub-set in order to be as close as possible to the original training set. The bootstrapping part consists in the extraction of N samples for the training subset and M different samples for the validation subset. When we require multiple subset the samples are sampled with replacement, hence, in two subset we can find equal events. This stratified bootstrapping method allows us to have more stable results.
Note: the choice of the size of the training and validation sets depends on the number of signals belonging to the particular weight categories. Because of this particularity, in the code we used the value $n\_train = 152000$ and $n\_val = 95000$ to get a training and validation subset of 151998 and 94998 samples respectively; 3004 samples are therefore randomly dumped.

Finding the optimal number of neurons:

Here we consider the tuning of the number of hidden neurons of our Single Layer Neural Network.
The choice of the best number of neurons in the hidden layer is the usual trade-off : the more neurons, the more capacity of the NN but at the same time higher over-fitting. Thus, we compute 5 times the AMS on the validation dataset considering different number of neurons, successively we calculate mean and variance for each network configuration.
Figure 6 shows the average AMS and the standard deviation for different numbers of hidden neurons.
As we can see, the number of neurons in the hidden layer plays a huge role. A small number of neurons reduces the capacity of the network and hence it reduces the complexity of the output function. This help reduce the over-fitting but at the same time could under-fit the dataset. On the other hand, a large number of neurons produces more over-fitting that could be tackled using regularization. A very large number of neurons also slows down the training process since the computationally complexity of the network increases.

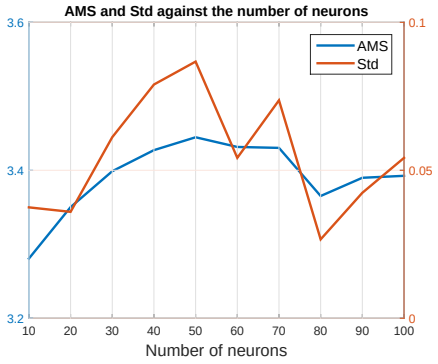


Figure 6: Average AMS and Standard Deviation of the NN for different number of neurons

A very large number of neurons also slows down the training process since the computationally complexity of the network increases. The choice of the number of neurons should not under-fit the data and at the same time we want a rapid training process. For this project, a good number of neurons is between 50 and 100.

5.1.2 Experimental results

Formally, the following table shows the maximum result we obtained during the validation process and on the official test set for a single layer Neural Network composed of 50 hidden neurons.

Network type/results	Validation AMS	Test AMS
Single Layer neural Network [30 50 1]	3.44	3.39

Table 2: Validation and test results considering a single layer NN

It is interesting to note the difference between the validation AMS and the test AMS. After multiple trials we realize that there is always a gap between the validation AMS and the test AMS. Even if the split of the training dataset is done accurately using the stratification there is always an error. Averaging multiple validation AMS could probably reduce this gap with the test AMS but training multiple time a Neural Network with such huge dataset will just take too long.

5.2 Single Layer Neural Network + Missing Data

5.2.1 Implementation

In the dataset, as seen before, some data are corrupted and were replaced by -999. Until now, we did not take them into consideration, although this can have some influence on our prediction, especially for the calculation of the mean or the standard deviation.

In order to try to decrease the influence of these data, we have not considered them during the normalisation process. The missing data has been replaced initially with "NaN" and after the calculation of the mean and standard deviation these values have been set equal to zero in order to give them a lower importance. We can see on *Figure 7* the influence of handling the missing data on the validation AMS. These tests were made for different number of neurons to confirm our results analysed in the previous section. We observe that by handling the missing data, we get overall better result.
Therefore, this argues in favour of the relevance of taking this data into account. Another way can also be to try to predict the missing data by using for instance the distribution of this data or a Singular Value Decomposition approach.

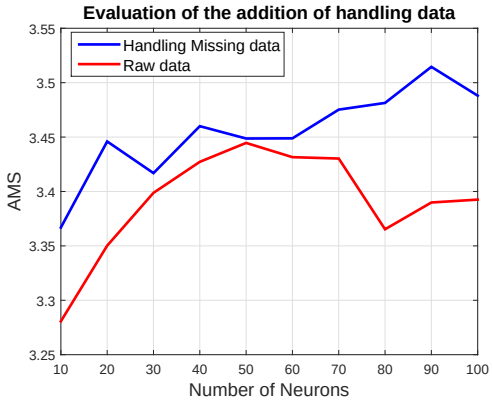


Figure 7: Validation AMS for different number of hidden neurons showing the influence of the missing data

5.2.2 Experimental results

Formally, the following table shows the result during the validation process and on the official test set for a single layer Neural Network composed of 50 and 80 hidden neurons by handling the missing data.

Network type/results	Validation AMS	Test AMS
Single Layer neural Network [30 50 1] + missing data	3.45	3.41
Single Layer neural Network [30 80 1] + missing data	3.48	3.44

Table 3: Validation and test results considering a single layer NN plus the missing data

As we can see, avoiding the missing data during the normalization produces overall better results than keeping the dataset unchanged. The difference between the NN [30 50 1] with and without missing data is not impressive however the difference becomes important when we considering other network with a different number of hidden neurons as shown in Figure 7.

5.3 Multiple Averaged Single Layer Neural Network + Missing Data

5.3.1 Averaging multiple networks

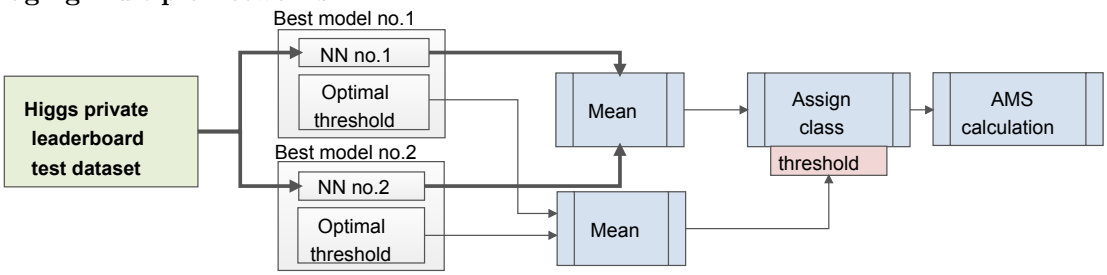


Figure 8: Example of averaging process for two trained Neural Networks

The diagram in figure 8 shows how two trained Neural Network are averaged. The models contain the optimal threshold extracted from the validation set and the respective weights and biases describing the optimal Neural Network. The test set passes through the two networks and the vectors, composed of floating values, are averaged. Successively, the resulting floating vector is compared to the average of the two optimal thresholds in order to define the class of belonging of each event.

There is another way to average multiple network as stated by Hinton [6] which consists in applying the geometrical average $\sqrt{NN_1 \cdot NN_2}$, however, in our case this has given equal results.

5.3.2 Experimental results

Network type/results	Number of NN averaged	Results of the NNs						Averaged result
		Validation AMS			Test AMS			Test AMS
Single Layer Neural Network [30 50 1]	5	3.34	3.38	3.41	3.34	3.34	3.40	3.46
		3.40	3.41		3.35	3.39		
Single Layer Neural Network [30 80 1]	5	3.53	3.54	3.44	3.35	3.43	3.41	3.49
		3.50	3.34		3.39	3.45		

Table 4: Validation and test results of multiple averaged Single Layer Neural Networks

As we can see from Table 4, the five results of the Single Layer Neural Network on the test set are around 3.35 - 3.4 whereas the result after the averaging process is considerably improved, raising the AMS to around 3.46 using 50 hidden neurons and 3.49 using 80 hidden neurons. Another important consideration is about the over-fitting, with 80 neurons we started to see some over-fitting but it is still not significant enough to introduce some regularization methods.To conclude, averaging multiple prediction/hypothesis is worthy and improves considerably the final result.

5.4 Deep Neural Network + Missing Data

5.4.1 Implementation

This section aims to describe and explain the classification framework concerning the Deep Neural Network plus the handling of the missing data.

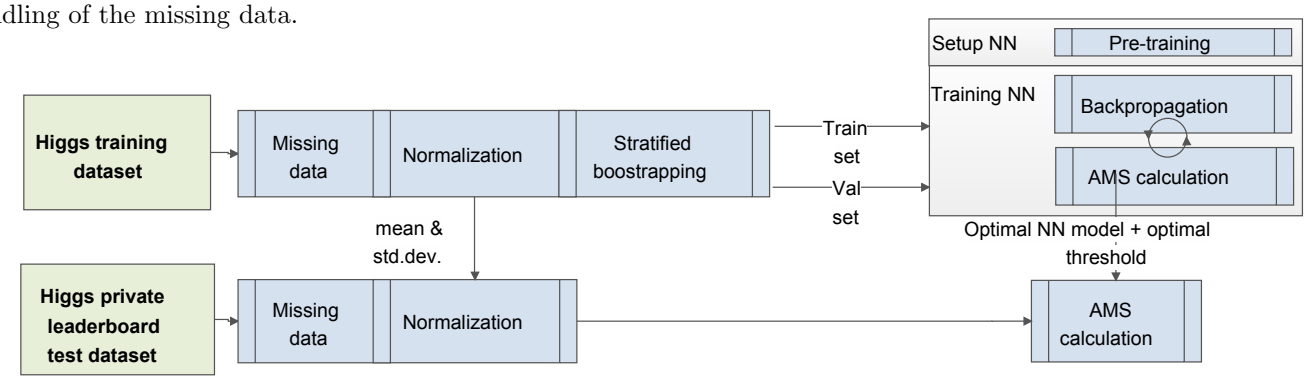


Figure 9: Deep Neural Network + handling of the missing data framework

Normalization: The normalization of the dataset is done considering the missing data as explained in the previous section and by using the classic zero mean unity standard deviation method.

The Deep Neural Network: There are basically two distinct ways to construct a Deep Neural Network (DNN), one consists in stacking different Restricted Boltzman Machine (RMB) and the second option consists in using the Auto-Encoders (AE). The AE is easier to implement and is also easier to understand, for these reason we used an ensemble of auto-encoder to construct the DNN. An auto-encoder is a simple Neural Network composed of one input layer, one hidden layer and one output layer. The input and output layers have the same number of neurons since we deal with an unsupervised problem. The mapping from the input to the hidden layer is the step that matter. We an AE, we are trying to transform the training set in another meaningful representation. This approach is similar to the feature extraction we apply to an image for instance in a scene recognition problem. The main problem of using a simple auto-encoder is that there are millions of way to set the weights in such way to have zero error between input and output and this is not different from putting the weights randomly. What we want is a meaningful representation of the dataset on the hidden layer. A denoising auto-encoder is a simple auto-encoder that stochastically sets some input neurons to zero as in the dropout method. This approach forces the auto-encoder to learn a meaningful way to map the dataset. The pre-training process trains each denoising auto-encoder separately using the hidden layer's output of the previously trained AE. The input-hidden weights are then stacked to form the final Deep Network.

5.4.2 Experimental Results

This section aims to compare the performance of Deep Neural networks with and without the unsupervised pre-training process, and to compare the DNN with the single layer NN for this specific challenge.
The first test consists in training a classic Neural Network composed of three hidden layers of 80 neurons ([30 80 80 80 1]) without the unsupervised pre-training process. The second test consist in stacking the pre-trained denoising autoencoders

before the final training of the whole network. For this last test, the percentage of the masked inputs in the denoising autoencoders has been set equal to 10% and 25% for comparison. The number of epochs is set to 50 for both pre-training and fine training. The learning rate equal to 1.5 and we used a mini-batch of 150 samples along with the sigmoid function as activation functions for the hidden layers and the output layers.

Network type/results	Validation AMS	Test AMS
Deep NN	3.52	3.50
Deep NN + pre-training AE denoising 10%	3.64	3.53
Deep NN + pre-training AE denoising 25%	3.49	3.49

Table 5: Validation and test results considering a DNN without pre-training and DNNs with pre-trained autoencoders

Table 5 shows the validation AMS extracted during the training process and the test AMS computed using the official test dataset. The Deep Neural Network trained directly performs unexpectedly well. Usually, a huge network is hard to train due to its depth. The backpropagation algorithm propagates the derivative of the activation functions from the output to the inputs, during this process the gradient gradually decreases along the hidden layers penalising the learning process of the bottom layers. The results of the DNN with pre-training process are slightly better to the DNN without pre-training. The pre-trained DNN with 10% noise produced the best result and will be used in the next section.

5.5 Multiple Averaged Deep Neural Network + Missing Data

5.5.1 Implementation

For averaging multiple Deep Neural Network outputs we simply used the same basic framework presented in Figure 8. Each DNN produces a floating prediction vector using the test set and this vectors are averaged. The optimal threshold is the average of the optimal thresholds calculated during the training process.

5.5.2 Experimental Results

Network type/results	Number of NN averaged	Results of the DNNs						Averaged result
		Validation AMS			Test AMS			Test AMS
Deep NN [30 80 80 80 1] with stacked AE denoising 10%	5	3.64	3.55	3.58	3.54	3.48	3.56	3.60
		3.54	3.68		3.48	3.54		
	15	3.48	3.68	3.43	3.48	3.52	3.47	3.64
		3.53	3.55	3.62	3.51	3.49	3.45	
		3.65	3.52	3.50	3.54	3.51	3.46	
		3.45	3.57	3.59	3.50	3.53	3.46	
		3.60	3.50	3.49	3.50	3.53	3.51	

Table 6: Validation and test results of multiple averaged DNNs with pre-training

Table 6 shows the validation AMS and test AMS for each Deep Network and the resulting AMS after the averaging process. The architecture of the DNN is the same as before, 3 hidden layer composed of 80 neurons each. As we can see, the overal result of a Deep Network is around 3.52 whereas the result after the averaging process exceeds 3.6 when we averaged 15 Deep Neural Network.

5.6 Multiple Averaged Deep Neural Network + Missing Data + Error weights

5.6.1 Weighted error function

Thus far, we used the classic Gradient Descent algorithm which defines the quadratic cost function as the sum of the squared errors (2):

$$E = \frac{1}{2}\sum_n (\hat{y}_n - t_n)^2 \qquad (2) \qquad\qquad E = \frac{1}{2}\sum_n (\hat{y}_n - t_n)^2 \cdot w_n \qquad (3)$$

where n is the sample number.
For a dataset composed of equally weighted samples, the classic cost function is the optimal solution. However, in the Higgs Boson dataset each sample is associated to a weight defining it's importance.In order to take advantage of this weights we used the weighted error function (3) as cost function to minimize during the training process.
Such cost function forces the network to learn more when the sample we are using is more important.

5.6.2 Choice of error weights

The error weights are set according to the weights of the training dataset. Since we do not know how to adapt the training weights, we experimented several approaches as shrinking them from 0 to 1 or by putting them Gaussian distributed around 1 etc. After several experiment, we decided to keep the weights of the background events as they are and we amplified the weights of the signal events using a simple variable α.

$$error\_weight_n = \begin{cases} training\_weight_n \cdot \alpha & if \quad event = signal \\ training\_weight_n & if \quad event = background \end{cases} \tag{4}$$

In order to chose the best parameter α we set-up a basic single layer neural network [30 15 1] and by sweeping α we observed the AMS curves for the training set and validation set. The accuracy upper bound is always given by the training result. If we are not even able to reach for instance 3.8 AMS on the training set it is impossible to reach more on the validation or test sets. We looked therefore for the parameter α that rise the most the training AMS (Orange curve). The over-fitting will be tackled in the final algorithm.

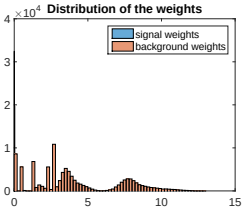


Figure 10: Distribution of the original training weights

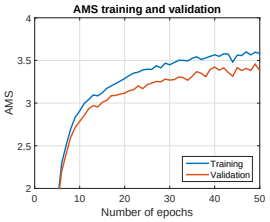


Figure 11: Validation and training AMS using $\alpha = 10$

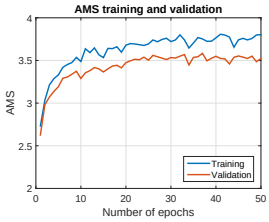


Figure 12: Validation and training AMS using $\alpha = 40$

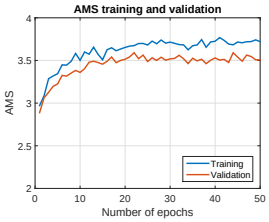


Figure 13: Validation and training AMS using $\alpha = 60$

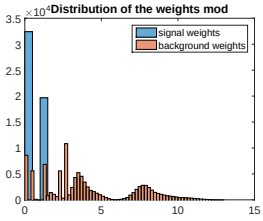


Figure 14: Distribution of the training weights using $\alpha = 40$

After several experiments using the basic Neural Network [30 15 1], the best value for α producing the bigger over-fitting is around 40. We also switched on the error weights only after the 10th iteration because of evident problems in the backpropagation process (see Figure 10).

5.6.3 Experimental Results

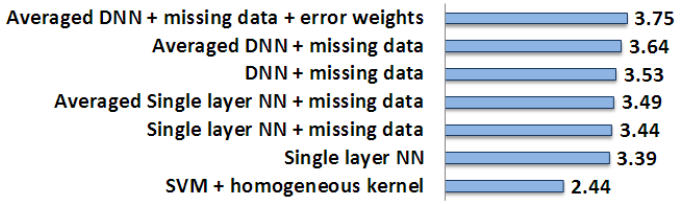
Network type/results	L2 weights decay	Number of NN averaged	Results of the DNNs						Averaged result
			Validation AMS			Test AMS			Test AMS
Deep NN [30 80 80 80 1] stacked AE denoising 10%	8e-6	5	3.82	3.76	3.78	3.70	3.65	3.68	3.73
			3.60	3.86		3.61	3.62		
	1e-5	5	3.64	3.87	3.70	3.67	3.67	3.62	3.75
			3.70	3.75		3.64	3.72		
	2e-5	5	3.68	3.66	3.65	3.60	3.65	3.62	3.69
			3.76	3.67		3.66	3.60		

Table 7: Validation and test results of multiple averaged DNNs with pre-training and error weights

Table 7 shows the validation and test AMS of each DNN and the resulting AMS after the averaging process. Each Deep Network produce now a very high AMS of around 3.65 and the best result after the averaging process is 3.75. In this case we used the L2 weights decay regularization method in order to generalize the resulting network.

6 Summary of Results and Discussion

The Support Vector Machine is a really powerful non parametric modal that always seek to the optimal solution, however this characteristic is really expensive for large scale datasets since the whole dataset is taken into account when we look for the support vector. The use of kernels improved considerably the final results but again the difficulties of implementing a true non-linear kernel has penalised this method.



The Neural Network seems being an appropriate model for solving high non-linear problems on huge datasets. From the beginning, a Single Layer Neural Network produced unexpectedly good results beating without problem the SVM. The improvement we made have increased constantly the final results especially the Deep network, the averaging process and the error weights. A negative aspect of this model is the huge amount of parameters a Neural Networks has. Since finding the optimal value for every parameter is an infinite task, setting a Neural Network is somehow an art and need a lot of experience.
Our final results is far from what we expected at the beginning, the 3.75 AMS brought our solution to the 14th rank in the Kaggle Higgs Boson competition against 1800 teams. Although the competition is no longer active, this result is anyway valid and very satisfactory for us.

7 Conclusions

During this project we applied our knowledges in machine learning to solve a complex and very interesting binary problem provided by the CERN in the form of a Kaggle competition [1]. The main objectives for this project were implementing different classifiers, improving and tuning them and finally compare virtue and vices of each method. For each classifiers we faced several problem tie with the size of the dataset. The amount of time we needed to train a classifier was a difficulties along as the weights associated to the dataset. The official metric used in this competition was the Approximate Median Significance (AMS) which depend on the magnitude of the weights and in turn to the number of samples. Splitting the dataset means reducing the number of samples in each subset and also means that the weights must change in amplitude in order to have correct AMS results. This proportion was not well specified in the description of the competition causing difficulties at the beginning of this project. To conclude, we achieved all the objectives specified from departure and the final AMS result obtained is an excellent outcome for us bringing our team to the 14th position in the ranking.

7.1 Future works

The number of possible improvement in our algorithms are still copious. The cross-entropy is another way to define the gradient descent cost function against the classic quadratic cost function. This modification is well known to speed up the convergence of Neural Networks and the precision in seeking the absolute minimum of the cost function. Improving the stratification when we split the training dataset is also to consider, especially the stratification of the background signal, thus far we stratified the signal only. In the Deep Network, it will be interesting to use the Restricted Boltzmann Machine instead of the Auto-encoders and compare the two solutions. Finally, trying to predict the missing values could also be an interesting task in a future project.

References

[1] Official challenge web page, `https://www.kaggle.com/c/higgs-boson`

[2] Official documentation and starter kits, `http://higgsml.lal.in2p3.fr/`

[3] Claire Adam-Bourdarios, Glen Cowan, Ccile Germain, Isabelle Guyon, Balzs Kgl, David Rousseau, *Learning to discover: the Higgs boson machine learning challenge*, `http://higgsml.lal.in2p3.fr/files/2014/04/documentation_v1.8.pdf`, 21 July 2014, version 1.8

[4] VLFeat web documentation API, `http://www.vlfeat.org/api/index.html`

[5] Andrea Vedaldi, *Efficient Additive Kernels via Explicit Feature Maps*, IEEE transactions on pattern analysis and machine learning intelligence, June 2011

[6] G. Hinton, *Lecture 6a: Convolutional neural networks for hand-written digit recognition* CSC2535: Advanced Machine Learning

[7] Rasmus Berg Palm, *Deep Learning Toolbox* , `http://www.mathworks.com/matlabcentral/fileexchange/38310-deep-learning-toolbox`

8 Students contributions

All four student who have participated on this project have contributed on the implementation of the algorithms, in the redaction of the report and on the presentation. However, *Leonardo Citraro* has supervised all the task and contributed more for the success of this project.