Optimization Project - MiniBooNE Particle Identification

Raul SENA ROJAS

1. Presentation of the Dataset

1.1 General Overview

This dataset was obtained from the repository of the University of California UCI - Machine Learning Repository. The dataset is taken from the MiniBooNE experiment and is used to distinguish electron neutrinos (signal) from muon neutrinos (background). The characteristics are the following:

- 130,065 instances (samples).
- 50 attributes.
- 2 categories (electron and muon)

The MiniBooNE experiment at Fermilab is designed to observe neutrino oscillations, BooNE is an acronym for Booster Neutrino Experiment. When doing the experiment the neutrino beam consists mostly of muon neutrinos and that is why the data is unbalanced, but this is going to be shown later in the EDA section. The 50 attributes from the dataset are continuous variables that are measurements taken in the MiniBooNE detector.

The motivation for choosing this dataset is because I'm interested in applying Statistical Learning to Physics. Also, I wanted to try optimization algorithms like Gradient Descent with a ~big dataset to see how the theoretical concepts seen in class performed in practice. However, working with a matrix of dimension 50 * 130,065 resulted to be an unfeasible experiment with the computational resources that I have, that's the reason I performed others methods to obtain good results in this problem.

1.2 Exploratory Data Analysis

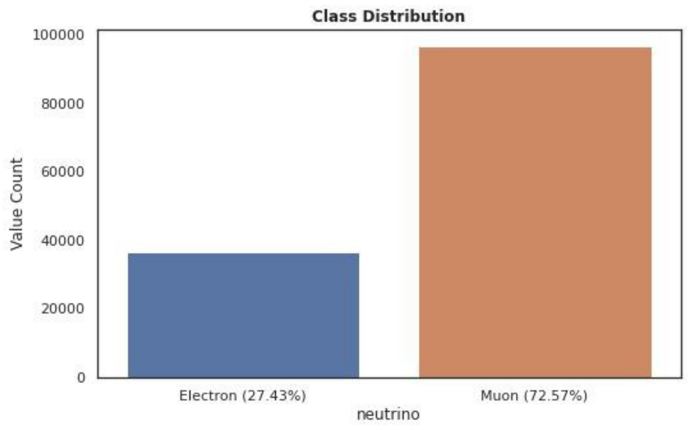


Figure 1 “Class Distribution”

The above figure shows that the data is Highly Imbalanced, because 72.57% of my data belongs to the class Muons, this is because in the BooNE experiment the neutrino beam consists of mostly Muons. There are many techniques I can use to overcome this obstacle like:

- Artificial Data Balancing: Down-sampling and Up-sampling techniques.
- Changing the loss function to take in to account the minority class.

The class distribution is important because with a very Unbalanced dataset I can’t really in metrics like accuracy to evaluate the optimal classifier.

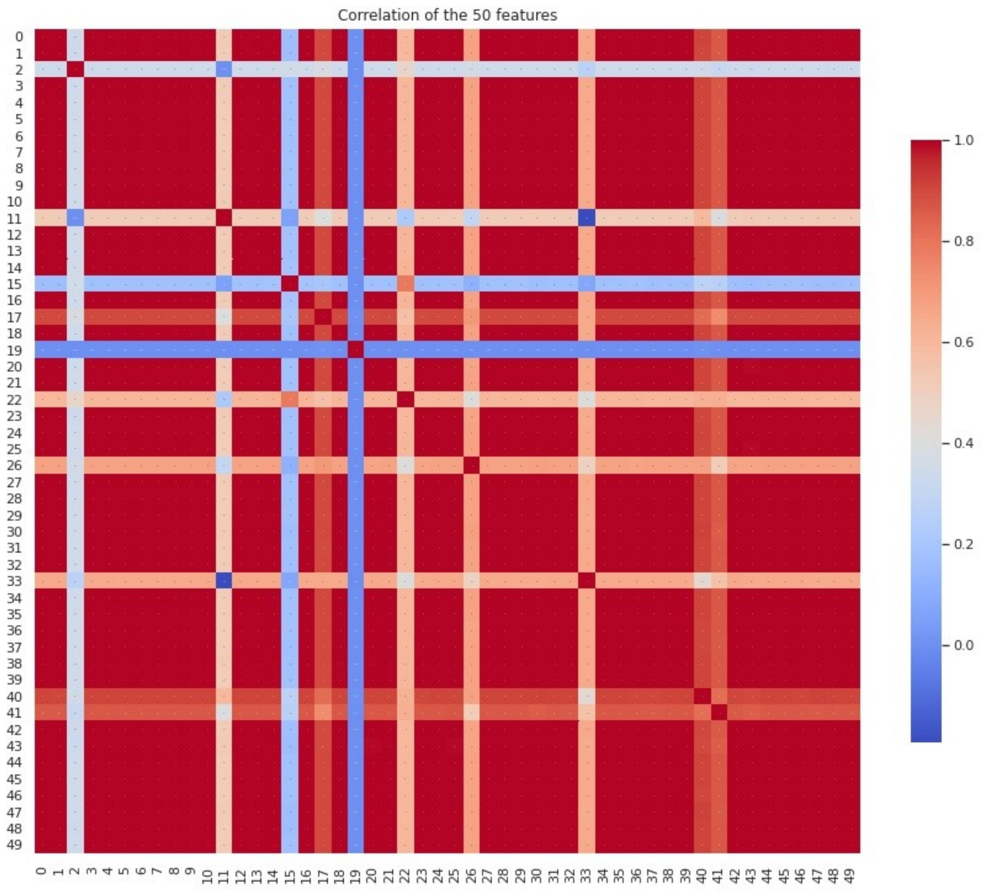


Figure 2 “Heat map of the correlations”

As we can see from Figure 2 the data is also highly correlated, which means there are some features that gives the same information, that’s why we are going to check different dimensional reductions so we can reduce the feature space. Additionally, it is important to mention that reducing the feature space dimensionality will help in the time of model computation and also will help in avoiding what some ml practitioners called “The curse of dimensionality”.

Before, we performed a dimensionality reduction technique we will scale our feature values with the MinMaxscaler of sklearn. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

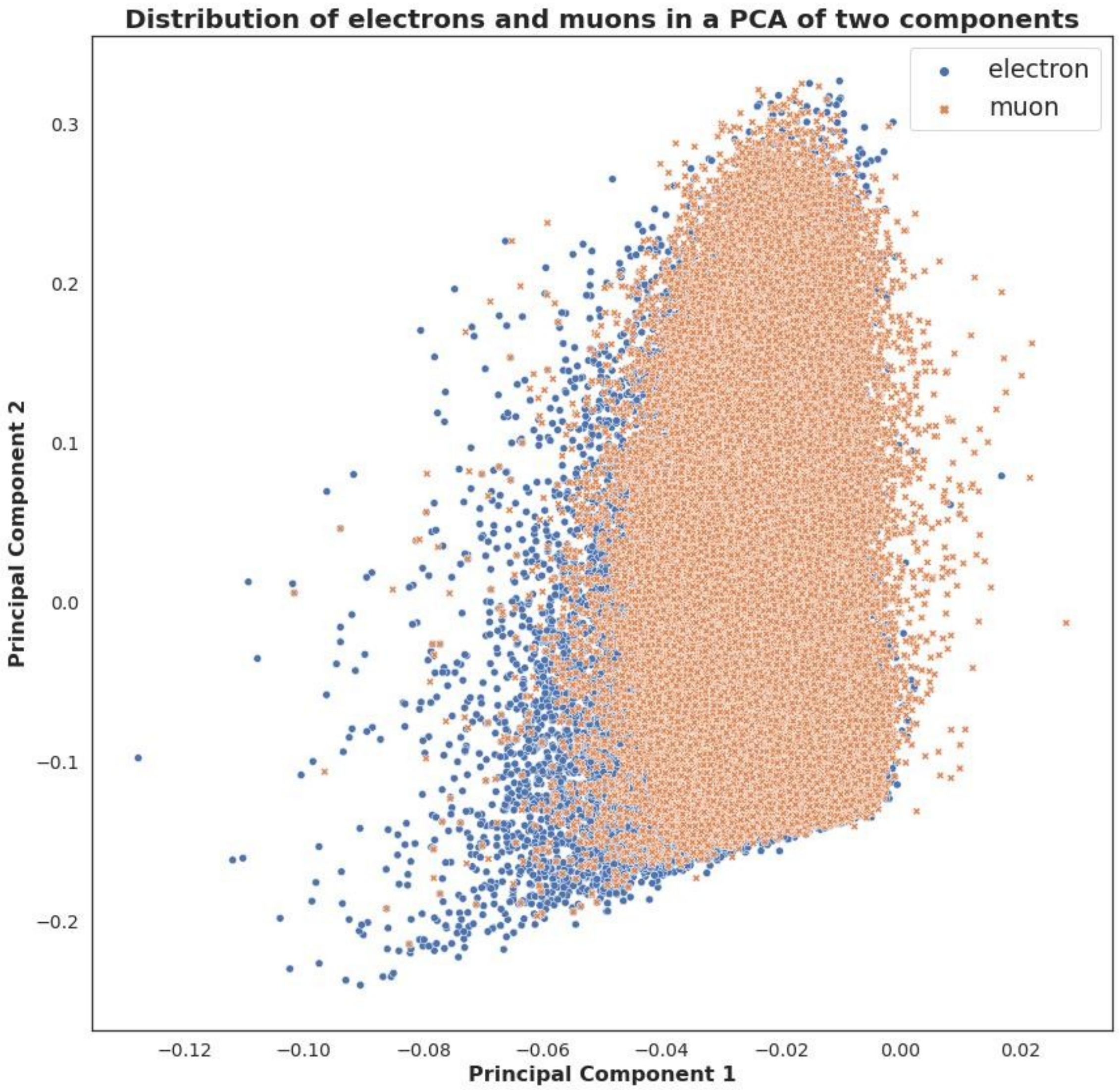


Figure 3 “Plotting the labels in a 2d space with PCA”

We can see in Figure 3 that as we expected the data is not linearly separable in 2d, that means that we will need more PCA components to perform logistic regression. However we can see that there are some electrons that are very far away from the cloud of muons, so they should be easy to classify.

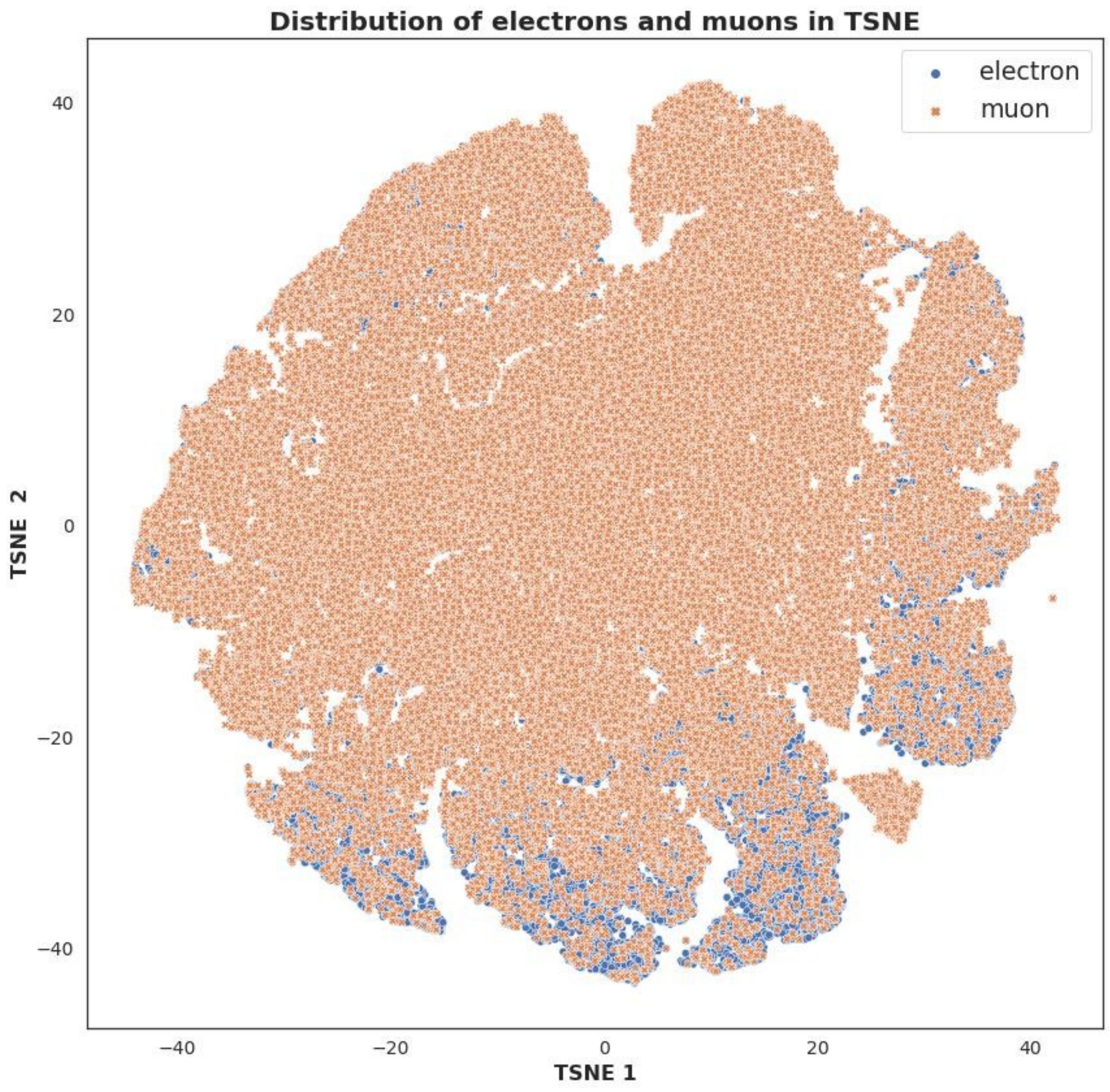


Figure 4 “Plotting the labels in a 2d space with TSNE”

We used another reduction algorithm named TSNE, one important thing here to mention is that the time to perform this algorithm was much greater than PCA. We can conclude from this plot the same statements as with PCA, the data is not linearly separable in 2d.

Which dimensionality reduction technique to choose? There are some theoretic intuitions on which dimensionality technique will be best. However, we are going to choose PCA because the computational time it takes to execute is much smaller than other techniques like: TSNE, ICA, etc.

However, in a real world problem will be good to create an iteration planned to choose the best reduction technique.

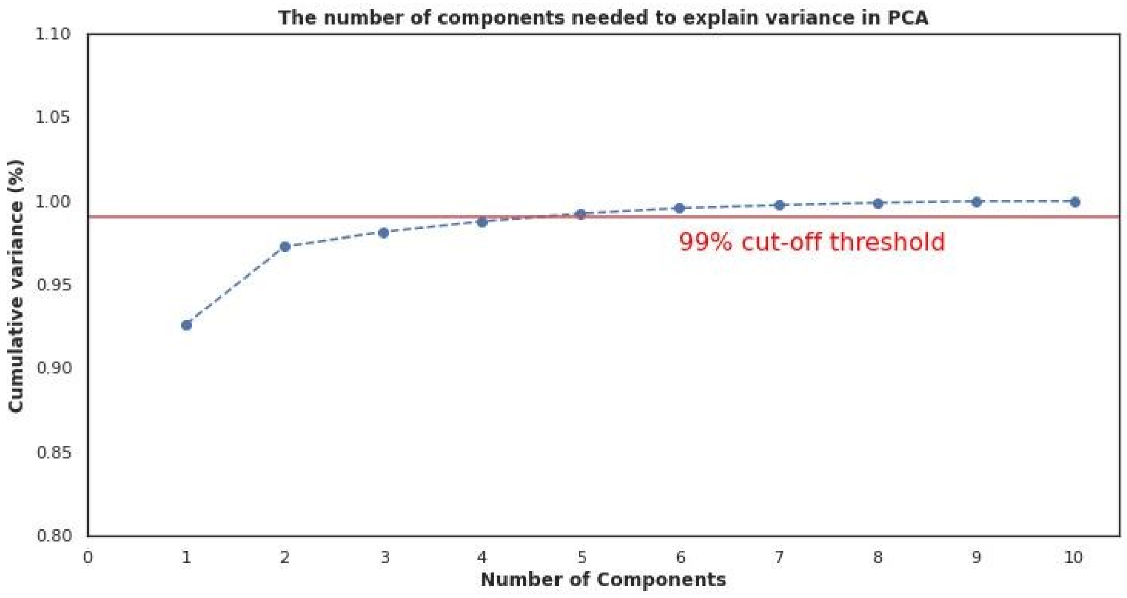


Figure 5 “Variance in PCA”

From figure 5 we can see that four or five components of PCA explained 99% of the variance of our problem that is why we are going to choose 5 PCA components.

Lastly, it is important to mention that we don’t find any Null values in the dataset and we did found 6 outliers, but they were removed from the dataset. Because we have almost 130k samples, removing the outliers will not affect our algorithm.

2. Presentation of the Problem

Our problem is a binary classification problem. That’s why we are going to use logistic regression, because it is a linear model, thus it is simpler and with a computational time that makes it feasible to train. However, due to it’s simplicity maybe we can’t obtain good results. The loss function better suited for Logistic Regression is:

$$\text{minimize}_{\mathbf{w}\in\mathbb{R}^d} f(w) := \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w}), \qquad f_i(\mathbf{w}) = \log(1+\exp(-y_i \mathbf{x}_i^T \mathbf{w})) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

$$y_i \text{ is a binary label in } \{-1, 1\} \text{ and } \lambda \geq 0$$

$$\mathbf{X} \in \mathbb{R}^{\mathbf{5}}$$

Equation 1 “Optimization Problem and Loss Function in Logistic Regression“

In our problem, the binary labels represent the following: -1 is the electrons and +1 the muons. It is important to mention that we are splitting our data into train and test set. X_train.shape = (93144, 5), X_test.shape = (39920, 5) and y_train = (93144,), y_test = (39920,)

3. Test of Stochastic Gradient Descent

In this section we will explained the experiments we performed with Stochastic Gradient Descent, as well as comparisons with Gradient Descent. The first thing that we learn at doing this project is that in some cases SGD is very slow and takes a lot of computational time.

The reason SGD is so slow is because with gradient descent, we calculate the gradient of all the dataset at once. With SGD, we calculate it on each sample, and then we do the same for other sample, until we have done 1 full pass through the data (1 epoch). In other words, in 1 epoch gradient descent makes only 1 gradient calculation and SGD makes n gradient calculations. Each calculation in SGD is faster than GD, but not n times faster. That’s why in this problem we couldn’t run the algorithms with the whole dataset ~93k samples in SGD, even though we tried. That’s why the following results with SGD are going to be of a sub sample of the previous dataset of 10k samples, in this sub sampling the data is going to be balanced, that means 5k samples of the category -1 and 5k samples of the category +1.

3.1 Different step sizes in Stochastic Gradient Descent

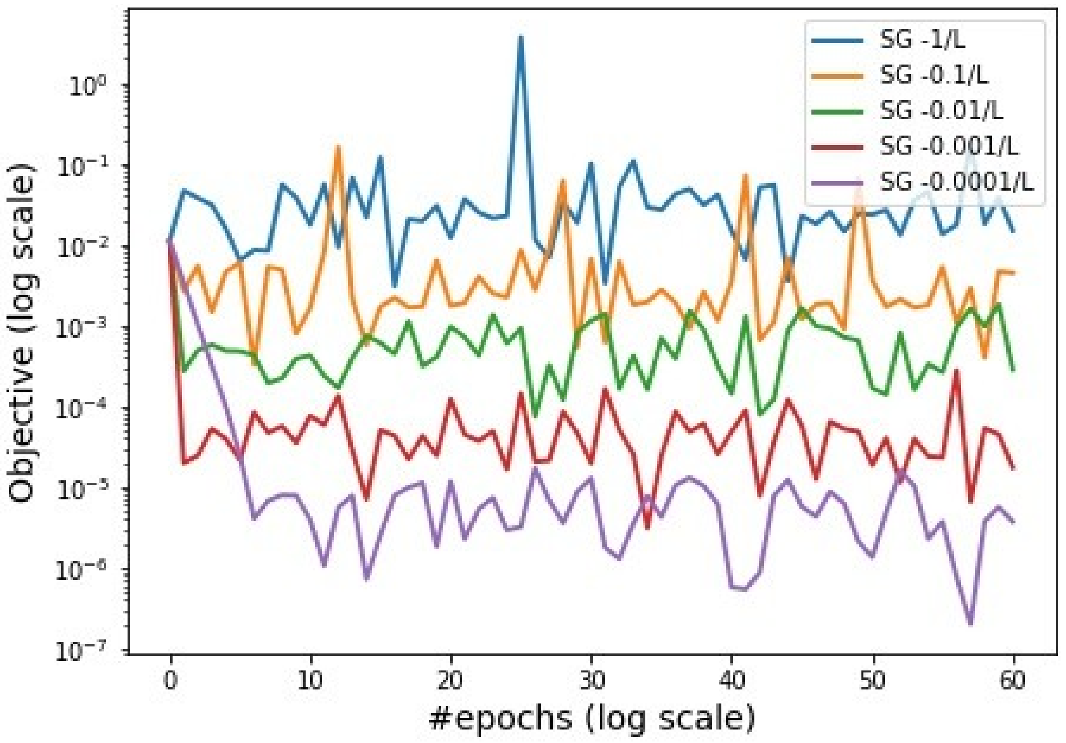


Figure 6 "Different Constant Step Sizes with Stochastic Gradient Descent"

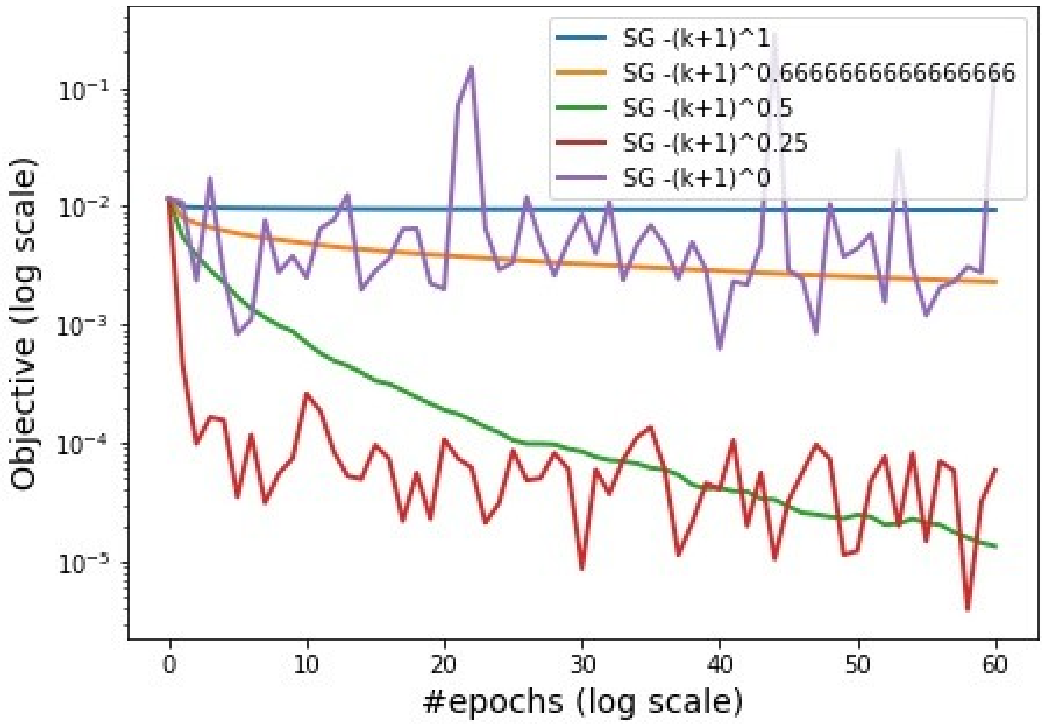


Figure 7"Different Decreasing Step Sizes with Stochastic Gradient Descent"

Observations of Figure 6 and 7:

1. **Both Figures:** We observe the oscillating phase of SGD, because the method stalls and takes steps that improve or worse the objective value.
2. **Both Figures:** We can see that the only step size that is actually converging is the decreasing step size of (k + 1) ^ 0.5. This is because there is no guarantee of converging with SGD, because it is not a descent method.
3. **Both Figures:** We can observe that SGD rapid decreases in the first iterations but then stalls and begin to bounce up and forward.
4. **Figure 6 (Constant Step Size):** It doesn't matter the step size, because neither of them are converging. Theoretically, using a very small step size will lead to slow convergence and a very big step size would lead to an oscillatory behavior with no convergence. That's the reason, that my guess is that all step sizes are too big and prevent SGD from converging. If I have to run this method again, I will choose even smaller step sizes.
5. **Figure 7 (Decreasing Step Size):** The blue and orange trends are trapped in a sub linear convergence rate. We can see the oscillatory behavior of SGD, as well as it rapid decreasing in the first iterations.

3.2 Different batch sizes in Stochastic Gradient Descent

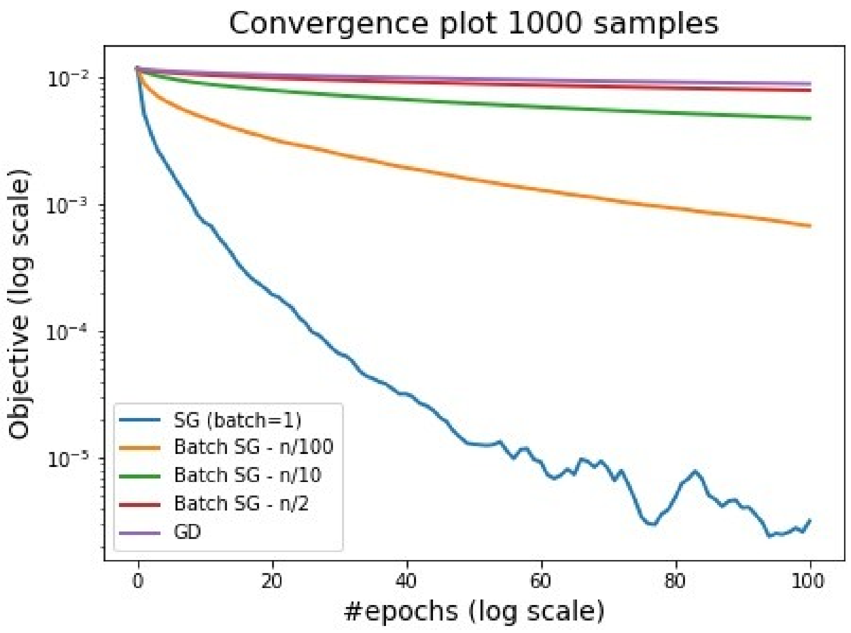


Figure 8 "Different Batch Sizes with Stochastic Gradient Descent"

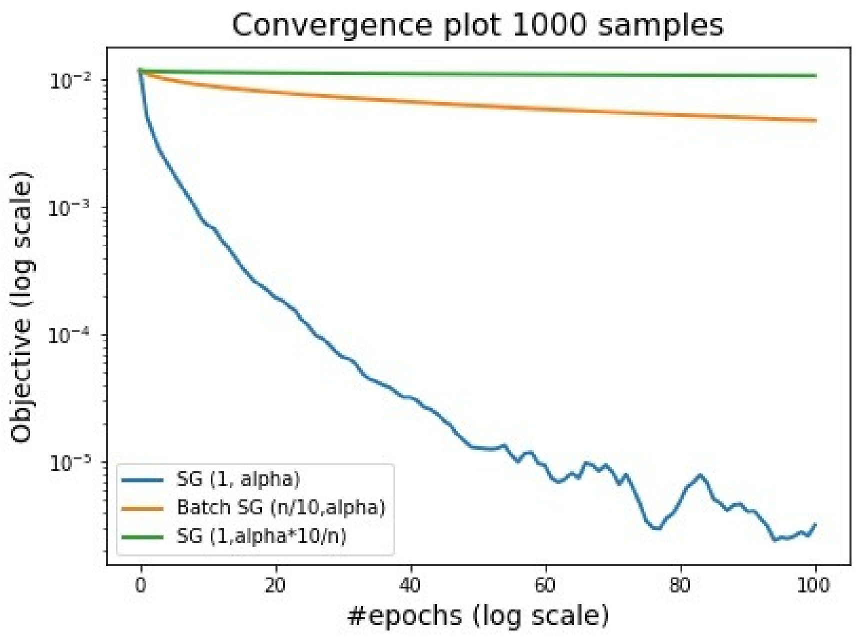


Figure 9 “Different Batch and Step Sizes with Stochastic Gradient Descent”

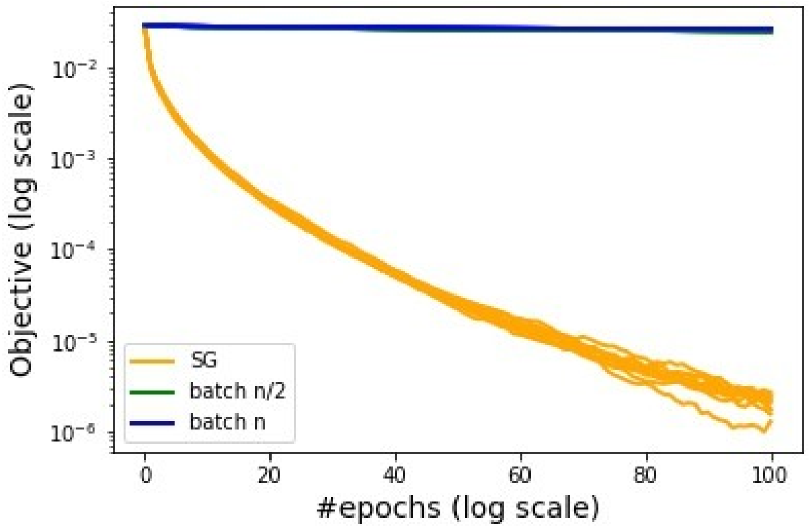


Figure 10 “Different Batch Sizes with 10 runs in Stochastic Gradient Descent”

To obtain the figures 8, 9 and 10 we had to sub sample the dataset of 10k samples to only 1k. The reason is that the computational time it took to compute the method SGD was too long.

Observations of Figures 8, 9 and 10:

1. In computational processing time the bigger the batch size, the faster it processes (When progressing the dataset of 10k).
2. **Both Figures**: We weren't expecting the behavior of SGD when the batch size is equal to 1, because it is decreasing in a good fashion and only at the end it presents and Oscillatory behavior. We taught that this behavior maybe was caused by a bug, but after the checking the code for many time we arrive at the following conclusion: The reason SGD batch size=1 is not presenting a very Oscillatory behavior is because in our 1k sample there is not a lot of variance in the data, so all the data is very similar and there is no outliers. However, a model trained with only 1k variable will obviously not generalize well the problem and perform poorly in test data.
3. **Figure 10 (Batch Sizes with several runs)**: We confirmed what we have said in point 1 and 2, because even though if we repeat the experiment multiples times the same results apply.

3.3 Stochastic Gradient Descent and Averaging

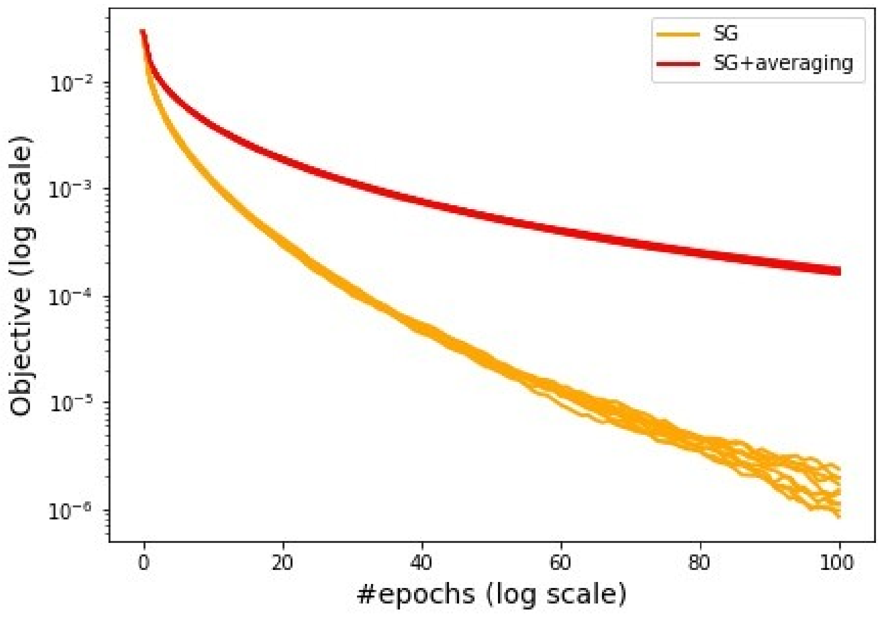


Figure 11 "SGD and averaging"

There is an error in the averaging of the plot in the Figure 11, the error is that it is averaging with an n values greater than the 1k samples we are using to generate the plot. It is only needed to move the red line below, However, we don't generate the plot again because it takes a lot of computational time and our observations wouldn't change.

Observations Figure 11:

1. One of the problem of SGD is that it has a very oscillating behavior, one of the methods we can use to avoid that kind of behavior is take the average of the iterate, because this leads to a smother behavior. In practice, this can work but also we have to take in to account the extra storage we need to store the vector of the average.
2. The averaging shows also how SGD begin to stall in some value after epoch 40, because as we have said previously there is no guarantee of converging with SGD.

3.4 Diagonal Scaling in SGD

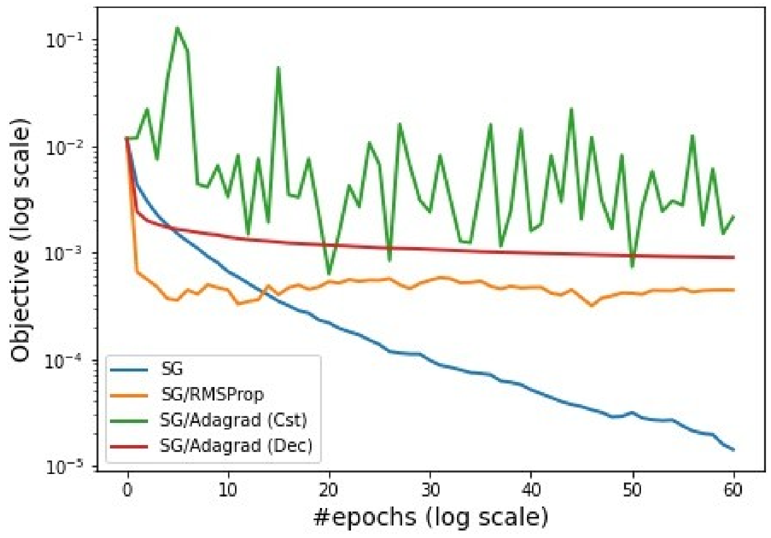


Figure 12 "Objective in Diagonal Scaling SGD"

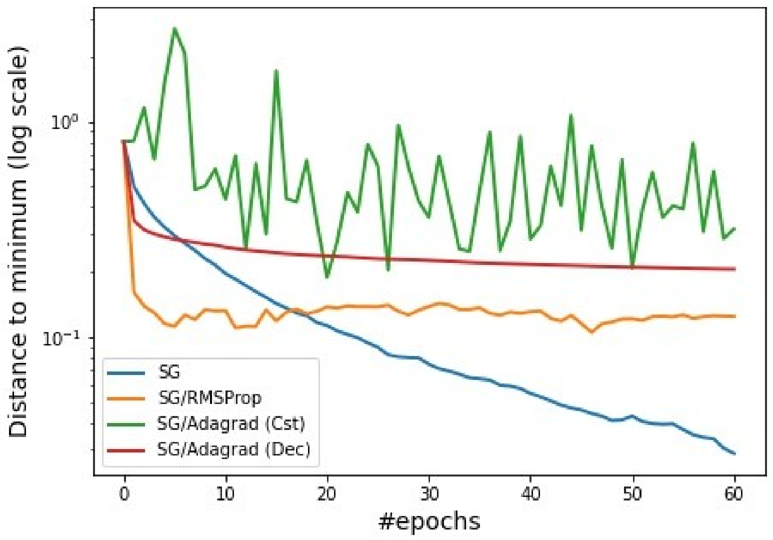


Figure 13 “Distance to minimum in Diagonal Scaling SGD”

Diagonal scaling of the gradient has make good progress with SGD and these ideas are implemented in RMSProp an Adagrad. We can see that in our problem, they don’t behave very well, because they stagnate in local minimum and the only method that is actually converging is normal SGD.

3.4 Stochastic Variance Reduced Gradient and Stochastic Average Gradient

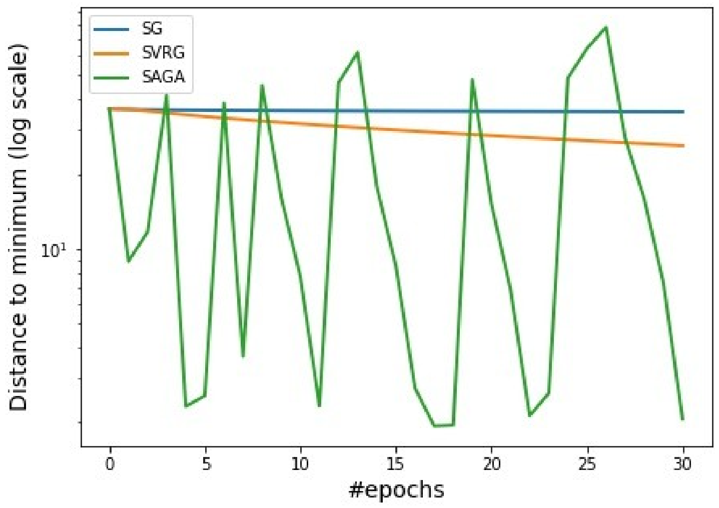


Figure 14 “SVRG, SAGA and SGD”

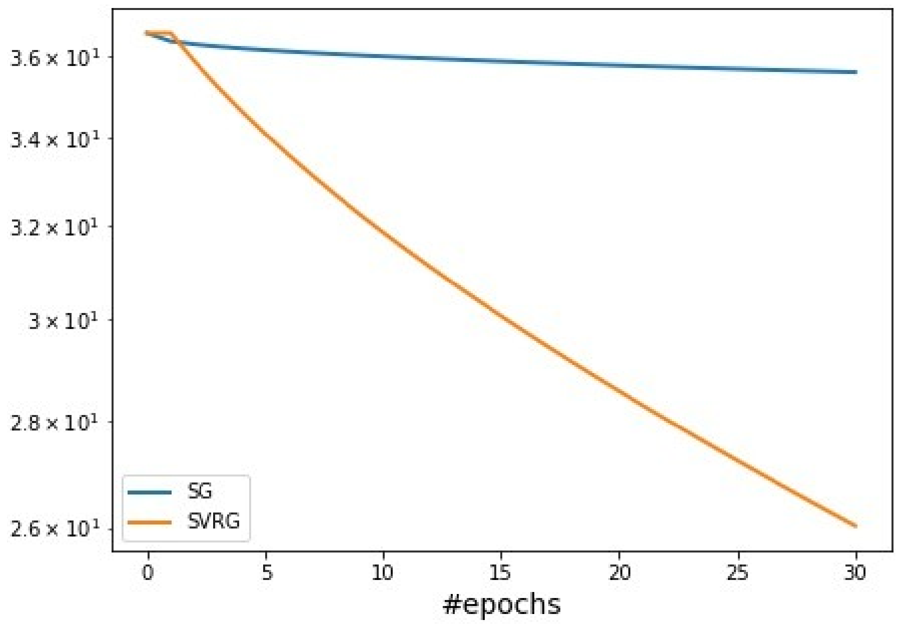


Figure 15

“SVRG and SGD zoom to Figure 14”

The great variation in SAGA in Figure 14 doesn’t allow to see how SVRG behaves. The bouncing in SAGA makes me believe that there is a bugged in my implementation of SAGA, and maybe it is the same bug of how I’m calculating the averaging. However, we can see how SVRG performs better than SG and this hadn’t happened in any other method.

4. Test of Regularization (with Gradient Descent)

4.1 Ridge Regression

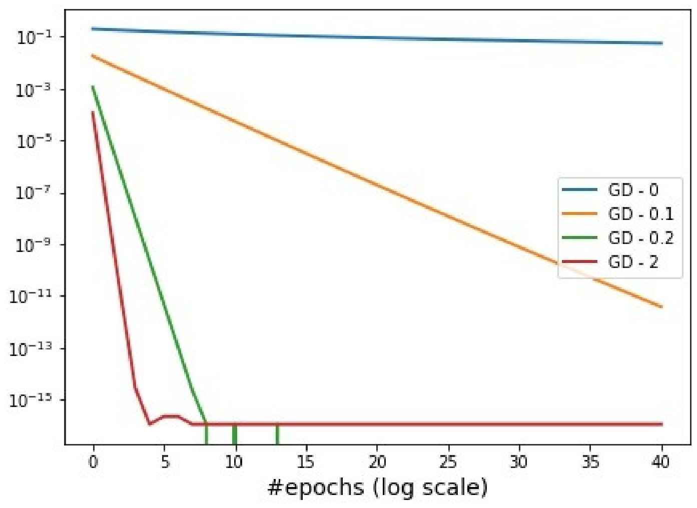


Figure 16 “Regularization using Ridge Regression”

4.2 Lasso Regression

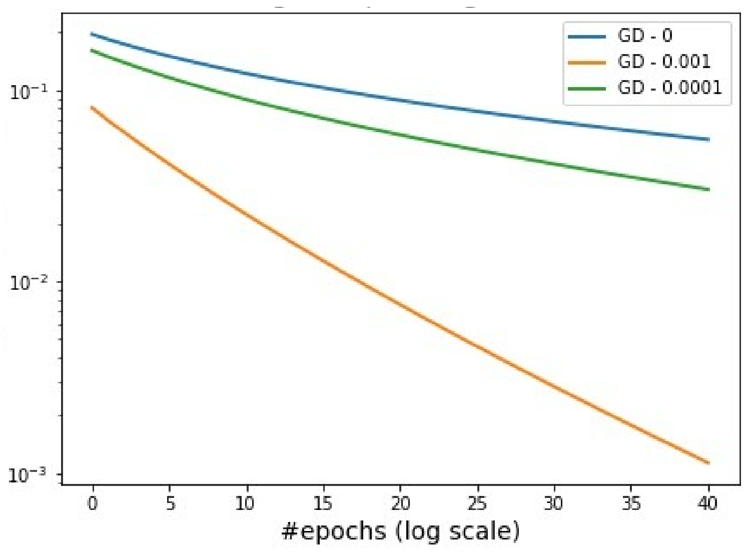


Figure 17 “Regularization using Lasso Regression”

The regularization idea is to limit the size of the weights W to make the model much simpler and being able to generalize better in the test data. The regularization parameter is added at the form of a sum in the loss function and the lambda hyper-parameter is tuned via cross validation. The difference between Ridge and Lasso is that in Ridge we want to optimize the square of the summatory of ws while in Lasso we want to minimize the absolute value of the summatory of ws.

Observations in Figure 16 and 17:

1. **Both Figures:** We can observe that the more we augment the hyper-parameter lambda the better, so in future iterations will be beneficial to try bigger lambda values.
2. **Both Figures:** Since the regression values are calculated with Gradient Descent we can observe the smooth descending curves, because in Gradient Descent we have guarantee of converging if the function we want to optimize is convex.
3. **Figure 16 (Ridge):** The model converge really quickly in the first iterations when using a lambda value of 0.2.

5. Test of Batch Gradient Descent

As we have explained in the introduction Batch Gradient Descent is much faster than Stochastic Gradient Descent and allows us to converge in to a result. All figures now on wards will be using all the training data, that is 93,144 samples.

5.1 Different Step Sizes Constant

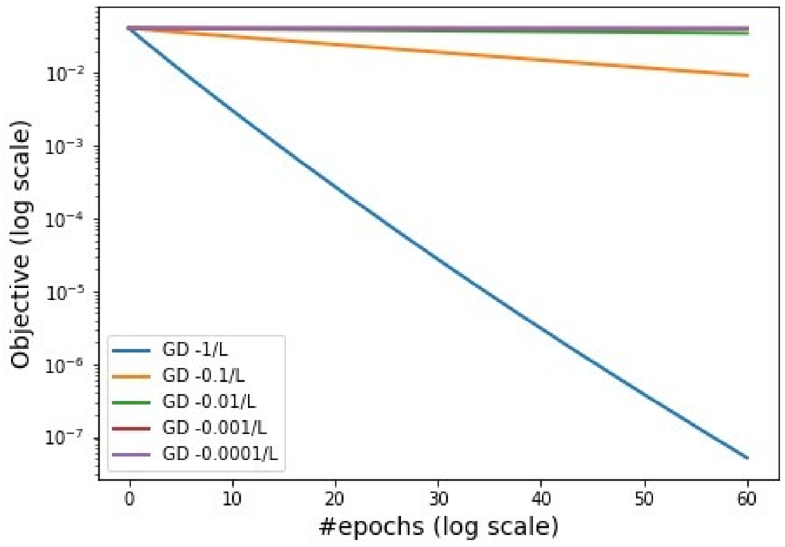


Figure 18 “Gradient Descent with different Constant Step Sizes”

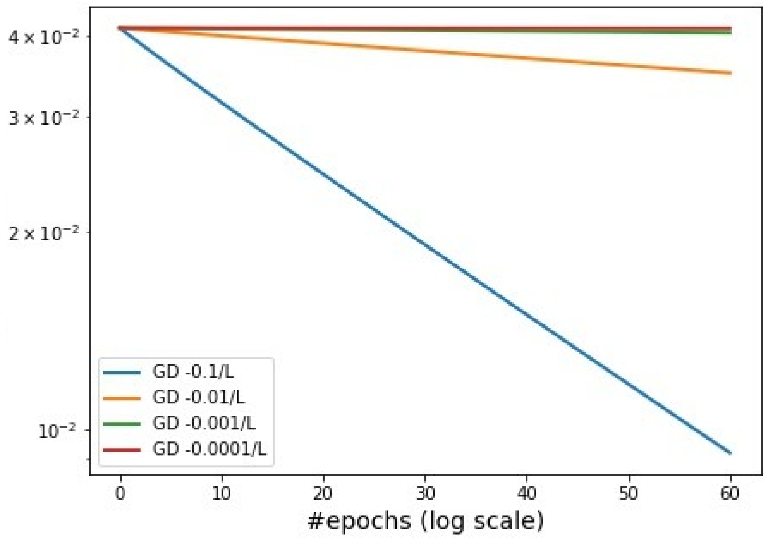


Figure 19 “Zoom to Figure 18”

Observations Figure 18 and 19:

1. **Both Figures:** The bigger the step size the better. This could be because the problem is very complex and in order to converge to a minimum we have to start with many epochs doing a big step size and then reducing it gradually. At the beginning w0 is very far from the optimum value so we have to do big step sizes to converge.

5.2 Different Step Sizes Decreasing

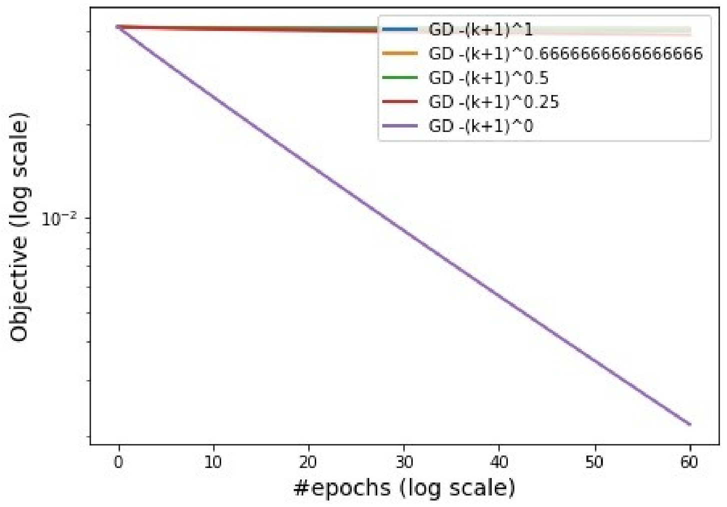


Figure 20 “Decreasing Step Sizes with Gradient Descent”

We get the same result as with the constant step sizes the bigger the step size the better. If I were to do more iterations of this problem I will probably begin with a bigger step size and eventually I will find a time when I need to decrease, but until now there’s no evidence that a smaller step size or decreasing it later will be beneficial.

6. Test of Accelerated Gradient Descent Nesterov

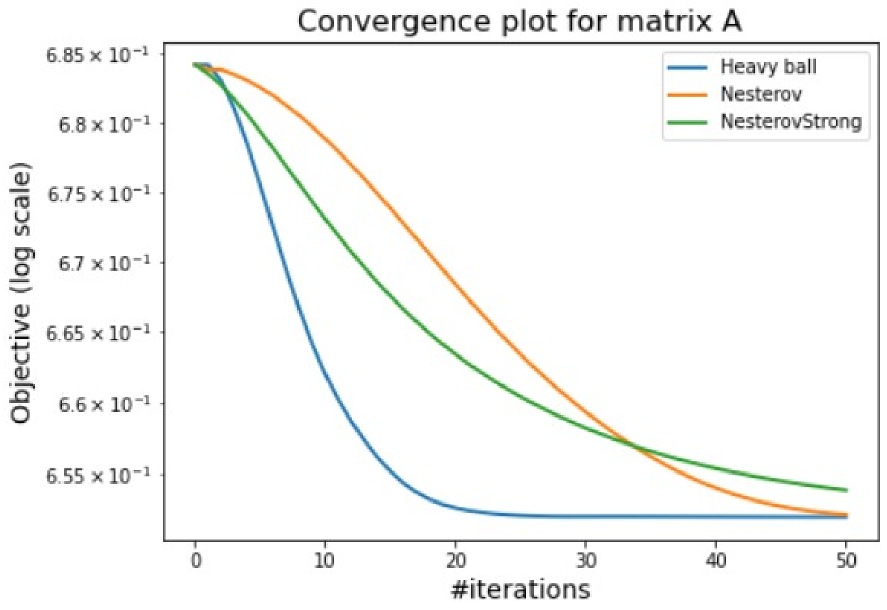


Figure 21 “Accelerated methods with Gradient Descent”

We can observe in the Figure 21 that the accelerated methods converge smoothly. We remove Gradient Descent from the graph, because Gradient Descent performs worse than this methods and that’s why it makes the graph not visible for the accelerated methods. In our dataset, all methods converges at the same object value in around 50 iterations. However, Heavy Ball reached that value in approximately 18 iterations.

7. General Conclusions

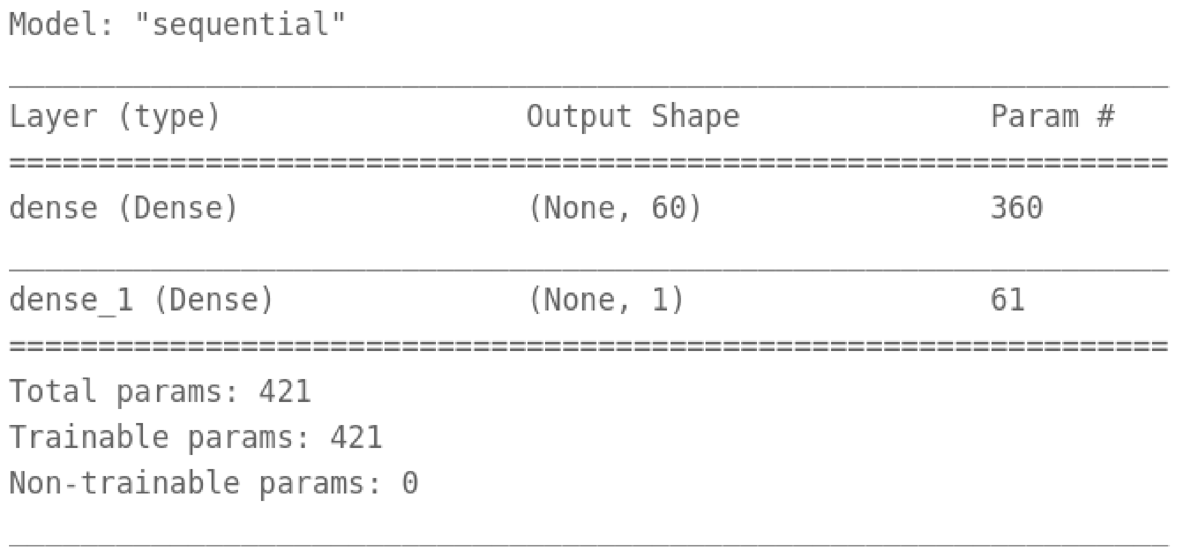
- In a problem with a descent size, for example this one with 130k samples, SGD doesn’t perform very well and also takes a lot of time to compute. If we add to the previous statement that there is no guarantee of converging, then it will be almost discarded from the toolbox when trying to solve a real world problem.
- It is always good practice to perform Regularization to be able to generalize well the problem.
- In the first epochs of a problem is good practice to plot learning curves to know if the learning rate is too small. We could see that because in some of our plots when doing a very small learning rate at the beginning we advance almost nothing in finding the optimal value.
- Even if SGD takes a small time in calculates the gradient, because if calculates in every sample, it will always last longer than Gradient Descent.
- The only type of problems were SGD will perform well is in those problems where there is no much variation in the data. We can see that when doing SGD for only 1k samples, because they were not so different SGD converges smoothly.

- SGD can get stuck in plateaus very often and that’s why sometimes have an oscillating behavior.

Optional Points

1. Test of more complicated models (Neural Network)

We trained the following basic Neural Networks with a binary cross entropy loss and a weighted accuracy metric. All with the 5 PCA components.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 60)                360
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 61
=================================================================
Total params: 421
Trainable params: 421
Non-trainable params: 0
_________________________________________________________________
```

Figure 1 “Basic Neural Network for Binary Classification”

The results were the following:

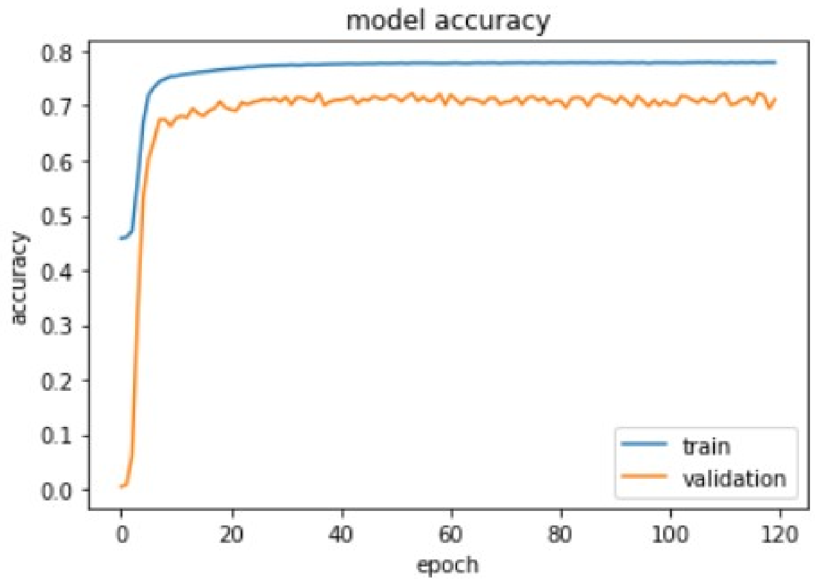


Figure 2 “Weighted Accuracy in a Basic Neural Network”

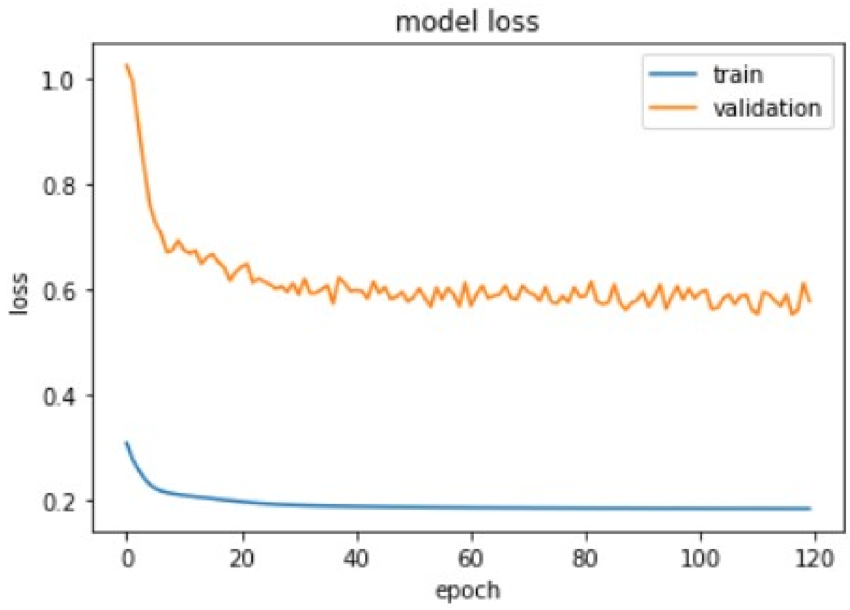


Figure 3 “Loss in a Basic Neural Network”

I’m quite surprise of seeing the Figure 2 and 3 from the history of the Neural Network. I taught the problem was more difficult and that it will require a more complicated model to achieve a good result. However we can see that the model generalizes the data very well in just a few epochs (taken in to account that we are calculating the weighted accuracy due to the imbalance of the data). However we are not able to surpass the limit of 0.8 in accuracy.

The impact of layers in train and test errors should be as follows: The more layers you add the better your model will be, but until a certain point. For example, if a classification problem is linearly separable, thus easy to solve a few layers will do good in train and test error, but a more complicated problem will need more layers. However, if you put many layers to a problem that is very simple is very likely that you will over fitted the data and due to that perform poorly in test data.

2. Use of early stopping and its impact on test error

As we can see in Figure 2 and 3 my problem doesn’t merit to do Early Stopping, maybe I need to add more layer to surpass the 0.8 in weighted accuracy. However, since this problem has already been solved and the best metric is around 0.9 I think it is very difficult to surpass the 0.8 in weighted accuracy.

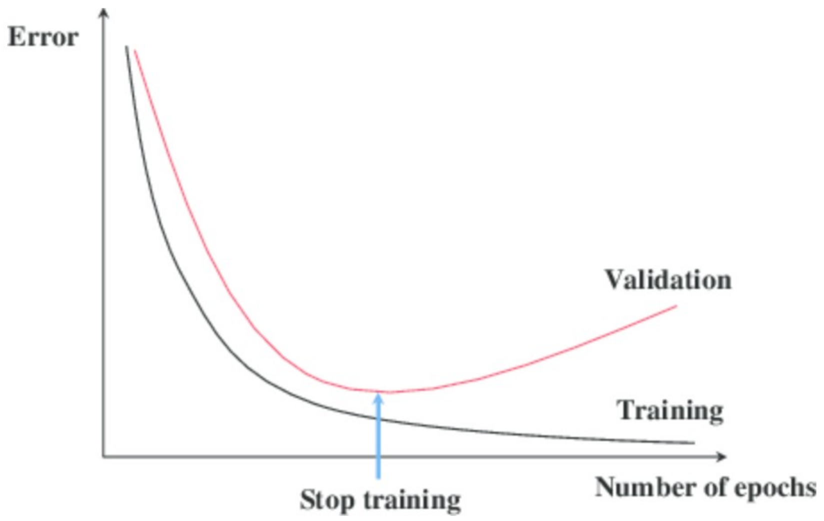


Figure 4 "When to use Early Stopping Theoretically"

As we can see in Figure 4, a model can perfectly learn all training data, but it will reach a point when it will begin to do overfitting. That is it fits so perfectly the training data, that it is not able to generalize it testing data. That's why we need to stop training when we see that the validation error begin to increase while the training error continue to decrease. In other words, performed well in the empirical risk minimization error, but bad in the true error risk.

3. Evaluate SVM

We trained a Support Vector Machine Classifier with a linear kernel and obtained the following results:

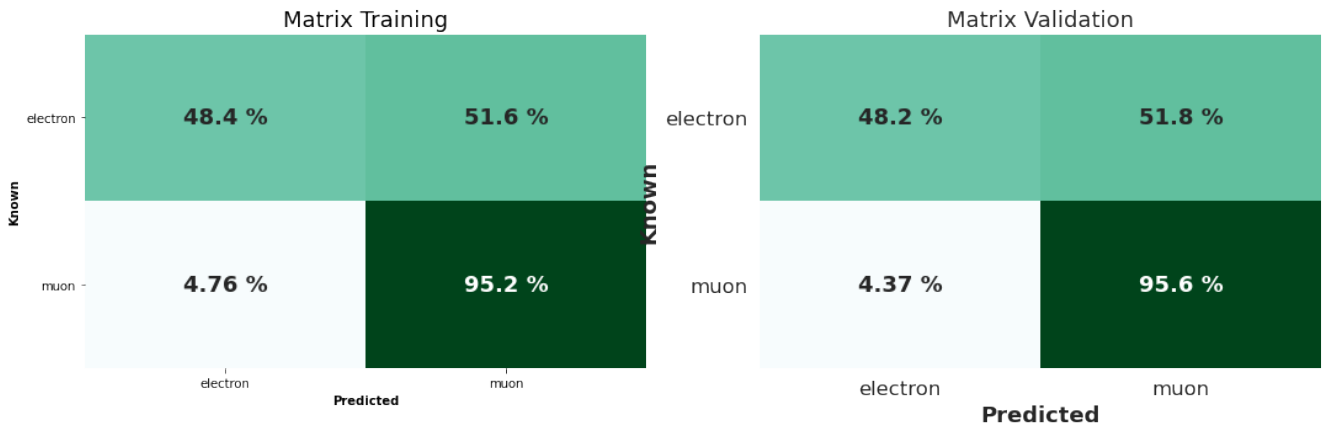
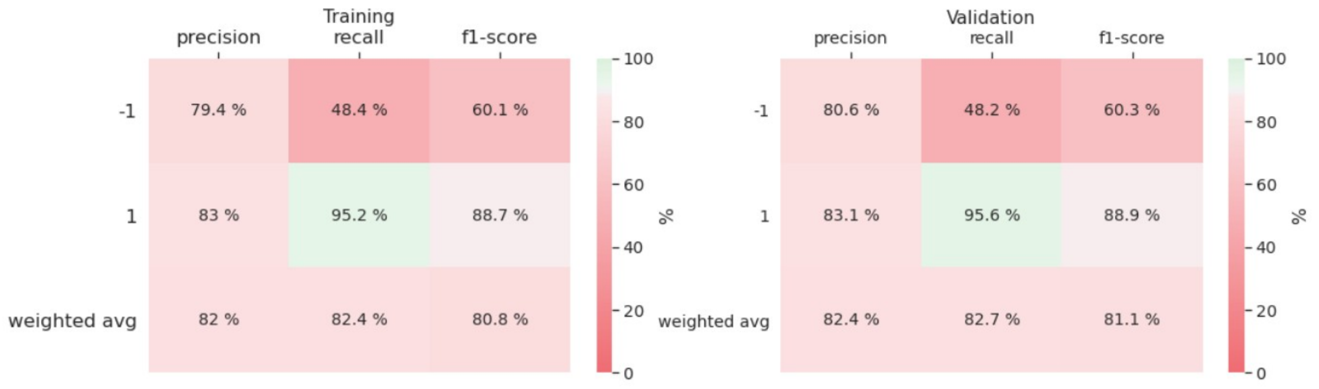


Figure 5 "Confusion Matrix SVM"

Thanks to the Confusion Matrix we can say the following, the SVM is not good and is not learning anything. We have to attack the problem of Data Imbalance, because our classifier is just predicting the majority class and that’s why it is not a good classifier.

Also I believed that my conclusions over the Neural Network are false, because there should be a bug problem that prevents the metric to weight the classes.



Training

	precision	recall	f1-score
-1	79.4 %	48.4 %	60.1 %
1	83 %	95.2 %	88.7 %
weighted avg	82 %	82.4 %	80.8 %

Validation

	precision	recall	f1-score
-1	80.6 %	48.2 %	60.3 %
1	83.1 %	95.6 %	88.9 %
weighted avg	82.4 %	82.7 %	81.1 %

Figure 6 “Classification Report in SVM”

As a reminder 1 corresponds to electrons and +1 to muons. We can see that in the classification report the model is not doing very well. Even if the weighted average is kind of high ~80% we can see in the confusion matrix that this can be misleading. Also if we see the f1 score of electrons (-1) it is very low, only 60.3%

In future iterations, it should be useful to plot this metrics with every model to compare them and to see how well they are doing and to do the necessary corrections in hyper-parameters, early stopping, regularization, etc. However, this metrics were not calculated for all models to make this more compact.

Note: In case there is any problem unzipping the code, you can check it in the following GitHub Repo: https://github.com/rsena-felipe/optimization_project

In that Git Hub repo will be a ReadMe.md that will explained how the code is organized.