DAT255 - Semester Project

Isabel Foster & Johan Bakøy

Search for new physics at the Large Hadron Collider

April 2022

In collaboration with the ATLAS research group at HVL

Contents

Chapter 1

Introduction

1.1 Goal and Motivation

There were three goals defined for this project:

- Create a classifier that can discriminate between SM physics processes and a particular extension of the SM (introduction of a new particle called Z').
- Measure the mass of the Z' particle in a simulated data set.
- Run the classifier on detector data to check for the presence of Z'.

These goals can be translated to:

- Create a binary classifier that classifies Signal from Background, where Signal is the new particle Z'.
- Calculate the mass of the predicted Signal.
- Run the classifier on unseen test data from the dataset.

The motivation for creating such a classifier, is mainly to see how Machine Learning and Deep Learning can help in the discovery of new physics. If it proves to be successful, there will be more alternatives for tools that physicists can use to improve our understanding of the world around us.

Potentially leading to answers to currently unanswered phenomenon, such as observations that are in disagreement with the SM e.g. the existence of dark matter.

1.2 Report Structure

In this report we will describe the project, it's importance and applications. We will also go through the process and discuss challenges and problems we encountered. Finally, we will present the results and evaluate to what extent we met the goals of the project, described in section 1.1

Chapter 2

Project Description

2.1 Project Context

This project lands in the broader context of Machine learning aided particle physics research. It's a fairly "simple" task as the data is well prepared for us, as well as similar previous works being available to learn from. The current research is a bit further than what the scope of our project is, so the insight learned in this project might not prove to be all that useful. Although as we will see later in this report our findings proves the consensus that tree-based models do very well on this type of data.

The project uses simulated detector data, as we can then label the data with far greater accuracy than real detector data. Hence we can train accurate models which we can then use on real detector data.

The datasets we've used have been constructed using a percentage of the whole dataset, to avoid issues with RAM (this is further explained in section 4.3), and to reduce the time required for training and fitting models. Because our goal is to classify new physics in the form of Z' particle (called "signal" in our project), we've chosen to do a binary classification, focusing only on determining if we have a signal (Z particle) and background (known particles in accordance with the standard model).

2.2 Related Work

This project borrows from the work of Dovydas Sprindys from last year's project[1]. While his work focused on feature selection, we focus on creating a classifier to separate new and old physics. We have used his work in reading and assembling the data, as well as the features he selected, with some additions of our own. Our goal has not been to recreate his work, but to use what he found and use that as a starting point, to build upon and improve where we can.

[1] https://github.com/dsp0011/DAT255-project/blob/main/ProjectReport-Final-v2.pdf

Chapter 3

Modelling

3.1 Baseline Model

We chose our baseline model to be a random forest. We did this for several reasons, but primarily because random forest generally performs quite well on tabular data. This became apparent very quickly, and we can see that the baseline performs very well in terms of accuracy.

```
Accuracy: 0.9953478710086537
```

This served as a motivation to create a deep learning model that would perform as well, or at least close to the same level of accuracy. While we did not achieve the former, our tabular models did get close.

3.2 Neural Networks

As we're using tabular data it makes sense to use the fastai tabular learner for this part of the project. We also use this as a baseline for the other deep learning models that we create. The base tabular model is a model with two hidden layers, dropout and embedding. It handles both categorical and numerical values.

As as starting point we constructed a tabular model using default parameters and and the original data (with normalization). The results were quite good already

epoch	train_loss	valid_loss	accuracy	f1_score	recall_score	precision_score	time
9	0.323645	0.323497	0.989619	0.674773	0.543448	0.889792	00:09

3.2.1 Loss Functions

We have tried some different loss functions to determine whether this has an effect on the models ability to separate new and old physics. Focal loss is a loss function that assigns harder cases greater weight, while the instances that are easy to classify gets assigned lower weights. This can be good for unbalanced datasets (as we have).

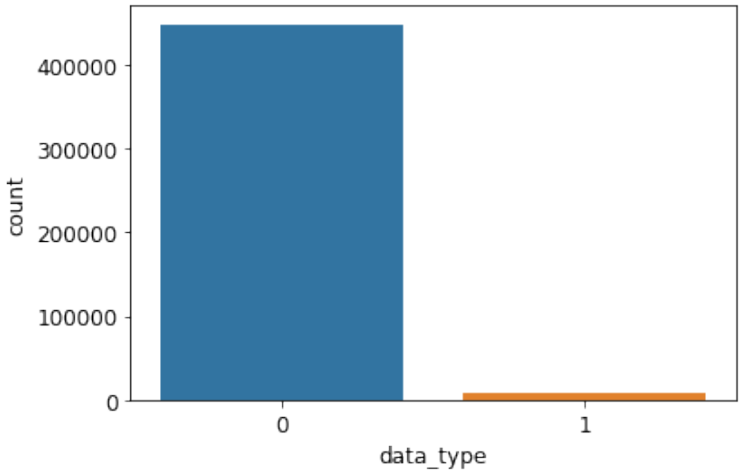
Cross Entropy loss is often used in classification tasks, and we are familiar with it from the course. We're also trying out a label smoothing cross entropy loss function that will not "punish" the model as hard for incorrect labels.

The accuracy doesn't improve a lot, the best loss function seems to be label smoothing cross entropy, getting marginally higher accuracy than focal loss and regular cross entropy.

epoch	train_loss	valid_loss	accuracy	f1_score	recall_score	precision_score	time
4	0.372871	0.372582	0.989526	0.666435	0.528000	0.903256	00:09

3.2.2 Under sampling

The dataset consists of a signal file and several other particle files. Becuase we're doing binary classification, the signal class is fairly underrepresented. The distribution of data can be seen in the figure below:



We therefore tried to adjust the class imbalance in the dataset by testing both under- and over sampling. For under sampling we tried three different strategies, first "not minority" that under-samples all but the minority class. In our case, because we are doing a binary classification this means we under sample the background. Second, we tried the sampling strategy "not majority", meaning we under sampled the signal, and finally we tried under sampling "all".

Both strategies "all" and "not minority" decreased the accuracy by quite a lot, ending up at around 88% after training for 5 epochs. The "not majority" strategy, however, performed much better.

epoch	train_loss	valid_loss	accuracy	f1_score	recall_score	precision_score	time
4	0.324200	0.323661	0.989449	0.663294	0.524414	0.902231	00:09

3.2.3 Over sampling

For over sampling we tried the sampling strategies "minority", "not minority" and "all". Again, the "not minority" strategy proved to produce the best accuracy.

epoch	train_loss	valid_loss	accuracy	f1_score	recall_score	precision_score	time
4	0.323984	0.323588	0.989542	0.671475	0.539310	0.889445	00:09

3.2.4 Layers and Embedding Size

The tabular learner has some parameters that we tried to utilize. We tried different combinations of layers, layer depth and embedding size. To avoid nesting for loops we used a library called optuna that constructs a study and attempts different combinations of the specified parameters. This process is really time consuming, so we only did this once in the early stages of the project, and used the results to optimize the model. The code is included in the file "Dat255_project_clean" [1], but not run there due to time constraints.

The results from previous runs showed, not surprisingly that more layers, and larger layers gave better results. After running all combinations of the provided values the best parameters proved to be a model with 4 layers of depth 800, 1000, 1200 (plus a final layer added by the tabular model), and embedding size of 0.1.

epoch	train_loss	valid_loss	accuracy	f1_score	recall_score	precision_score	time
9	0.324140	0.323653	0.989318	0.653792	0.508966	0.913819	02:25

This model likely over-fitted as the accuracy decreased and validation loss increased towards the end of the epochs.

[1] https://github.com/AHelplessStudent/Dat255_Course_Project/blob/main/DAT255_project_clean.ipynb

3.3 Discussion

Our chosen measurement is accuracy, but we still want to know more about what our models predict and where the mistakes are. Therefore we use a confusion matrix and a ROC plot to further assess the models. We will compare the baseline (random forest), and a model constructed from the most accurate attributes that were described in the sections 3.2.1 - 3.2.4 above.

One downfall of using accuracy as our metric, is the dataset being so unbalanced a classifier that only predicts background will still provide a high accuracy score. Therefore we tried mitigating this we over-sampling and other strategies, to see if they could benefit our models.

The baseline model performs better than our deep learning models. Our supervisors confirmed that this was expected because random forest models generally performs better on this type of data. Despite this we did manage to get quite close to the baseline accuracy, 99.53479 vs 98.93291

3.3.1 Confusion plots

A confusion matrix plot allows us to see what labels were misclassified by comparing the true label and the predicted label. We can use it to clearly see where our model makes mistakes. The plots can be seen in figure 3.1 and 3.2

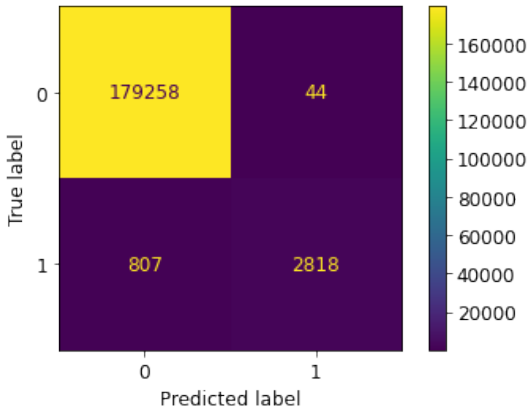


Figure 3.1: Random forest confusion plot

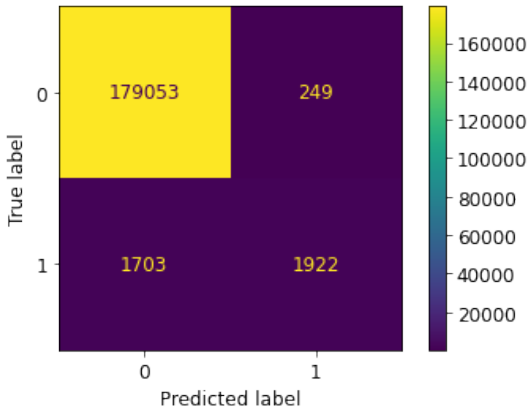


Figure 3.2: Tabular learner confusion plot

It's apparent from these plots that both models tends to make mistakes classifying signal (true label 1) as background.

3.3.2 ROC plots

A receiver operation characteristic curve is used to show the diagnostic ability of binary classifiers. An ideal model should be as close to the top left corner as possible. Again we can see, in figures 3.3 and 3.4, that the random forest model performs better. The ROC does not depend on class distribution, which makes it a good measure for imbalanced classes.

The tabular learner does have a false positive rate, and at some point delves under the 45° line, meaning at that point it actually performs worse than a random guess. This is despite the model having an accuracy of above 98%, showing how the AUC measure picks up on uneven class distributions.

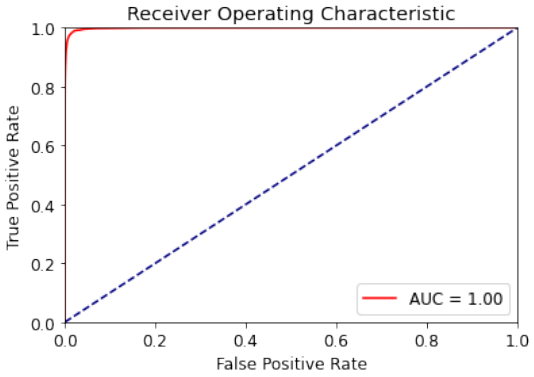


Figure 3.3: Random forest ROC plot

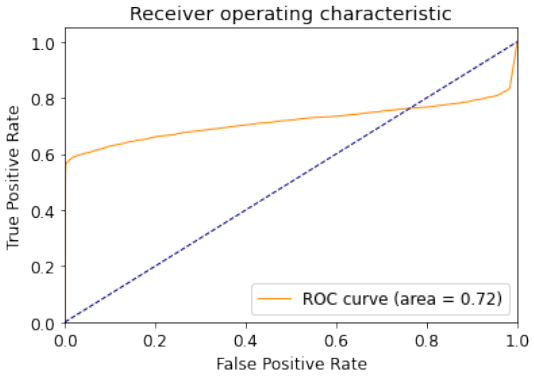
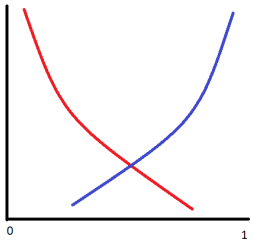


Figure 3.4: Tabular learner ROC plot

3.3.3 Signal and background plots

Finally, it's beneficial to plot the predicted signal vs background. Because of the properties of the features, signal and background, when plotted, should look something like this:



Where the background is red and signal is blue. From the predictions of our models we have constructed figure 3.5 and 3.6.

The plot shows the predicted values for the true signals (blue) and the true background (green). The random forest plot is much closer to what we would expect, and has a much smoother gradient. Also note that the axes of the plots are different as the tabular model didn't predict

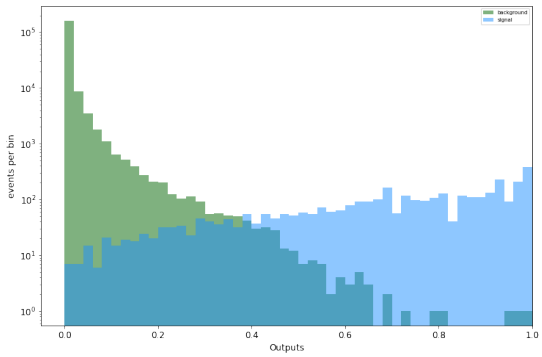


Figure 3.5: Random forest signal plot

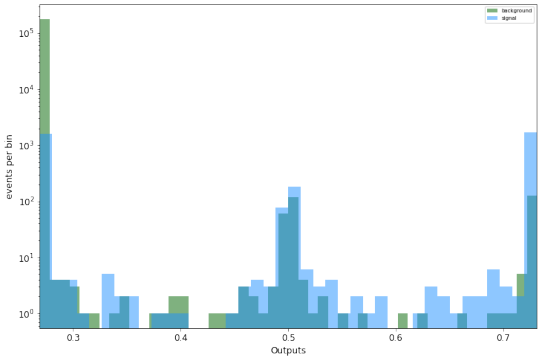


Figure 3.6: Tabular learner signal plot

anything above 0.75, meaning the model was never more than 75% certain an entry was a signal.

Chapter 4

Results

4.1 Mass results

Finally we calculate the mass of the predicted Signal, this was done by using new test set consisting of a larger data set, and using the best performing NN model on it. This left us with a dataframe with predicted and actual values for both signal and background. We then split it into predicted signal and predicted background. Given that the model predicted enough signal correctly the plot should show clear peaks above each Z' particle weight in the test set.

The mass was calculated using a formula given by our supervisor

$$M^2 = 2pT_1pT_2(\cosh\eta_1, \eta_2) - cos(\phi_1 - \phi_2))$$

As we are only interested in M we take the square root of the whole equation.

Since all our signal files are combined the peaks are fairly hard to spot, except the obvious ones on about 90000, and 15000.

We see the classifier predicted most of the signal correctly from how the predicted and actual plots have similar values, and they look roughly the same.

We were a bit late making these plots, so we could not get feedback on them, that they did in fact show what they needed to show. From our limited knowledge the peaks seem to be in the right spots. What might have made it clearer is to only predict on one type of signal file at a time, but as we were running low on time we decided to only predict on all the signal files

combined.

One observation that is somewhat obvious is the background jets being classified as signal causing a fair bit of noise in figure 4.3.

Log versions of these plots can be found in the Calc_mass notebook.

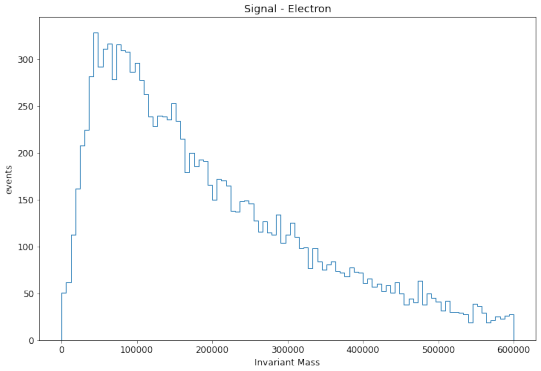


Figure 4.1: Lep - Predicted

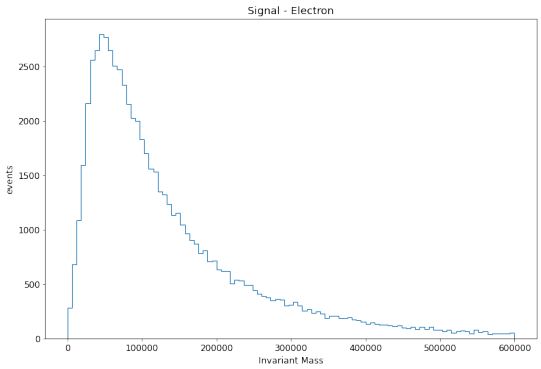


Figure 4.2: Lep - Actual

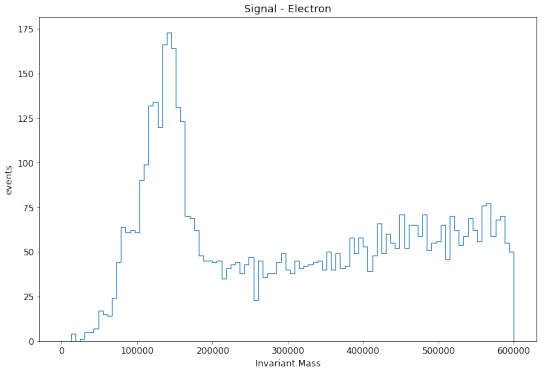


Figure 4.3: Jet - Predicted

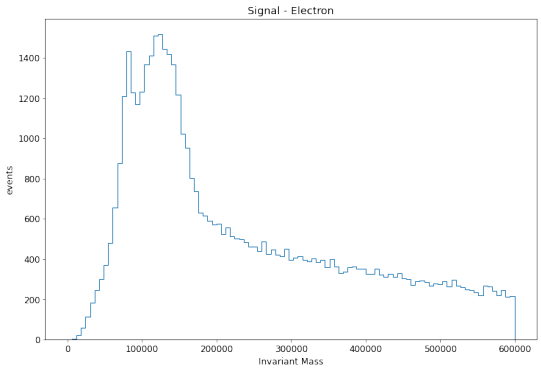


Figure 4.4: Jet - Actual

4.2 Discussion results

4.3 Issues

We got our dataset fairly late into the semester (towards the end of march), so we had less time to explore other solutions, as other classes were starting to ramp up with a heavier work load, around that time.

The main technical issue was our Ram being used up while using the Dataframes created with the entire dataset. The main way of solving this issue is of course using batching and only load parts of our data at a time. However due to the way our dataset was provided this proved to

be a fairly complex procedure and with the advice from our supervisor to just continue using a subset of the data, we chose to not look further into it. The solution we settled with was, as mentioned using a subset of the data. The code for this can be found in the "Data-prep" notebook. We had plans on using a larger subset of the data, but ended up just using 10%.

We also had trouble calculating the mass of the predicted Signal data, as we were selecting only some of the features needed to do so. We tried fixing this by including the features needed and excluding them from training. This was done to keep the feature list short while training, as we had previously chosen features that Dovydas had found were the most important through the feature selection he had done on the same dataset. This solution proved to be cumbersome to implement, so we chose the simpler option to just include the features in the training data. So our model trained with more features than we initially planned. This proved to have little to no effect on the classifiers we trained.

The issue of how to encode our data came up late, as we were unsure of how our choices would impact the underlying data, and the models ability to actually learn the patters in the data. So we settled on just using the mean normalization as our only main pre-processing, before feeding it into our models.