AMERICAN UNIVERSITY OF BEIRUT

CMPS 261

Project Report

Marc Mansour, 202102212

Jad Assaad, 202103426

Team Name: Yasser

1. Introduction

High-energy physics is the study of the fundamental properties of the physical universe by investigating the structure of matter and its interactions through the use of modern accelerators. These accelerators collide particles to create exotic particles that occur only at high-energy densities. The detection and measurement of these particles may provide insights into the nature of matter. Machine-learning approaches are often used to solve the difficult signal-versus-background classification problems in finding these rare particles. Good data analysis is crucial for distinguishing collisions that produce particles of interest (signal) from those producing other particles (background). Our task is a classification problem where the goal is to distinguish between a signal process where new theoretical Higgs bosons (HIGGS) are produced, and a background process with identical decay products but distinct kinematic features.

2. Problem Definition and Algorithm

2.1 Task Definition

The dataset consists of 28 input features and 1 output feature, from these 28 features, 21 were available directly and the other 7 were engineered. Our goal is to make the best use of these features so that we can maximize the accuracy when predicting the target value.

2.2 Algorithm Definition

Throughout this course, we have learned a variety of different algorithms and machine learning model, from linear regression to deep neural networks, we are able to test all of these acquired algorithms in our experiment to find out what model is best, keeping in mind that we still have to tune the hyperparameters of some of these models to maximize accuracy.

But before we start with testing models, we had to look at the data we have:

- First, we made sure that the output entries were distributed fairly, and indeed the number of 0s and 1s were more or less equal, and so the data was distributed fairly.
- Next, we checked if there were any missing values, and since we have 600,000 entries, dropping a few wouldn't hurt our model.
- After dropping a few missing entries, we decided to drop any duplicate entries because this could cause bias and damage our model's performance.
- We tried identifying outliers and removing them from the data, however this resulted in the deletion of half of the data, so we kept our data as last modified.
- Lastly, we split this data into three parts: Training 80%, Cross Validation 10% and Testing 10%

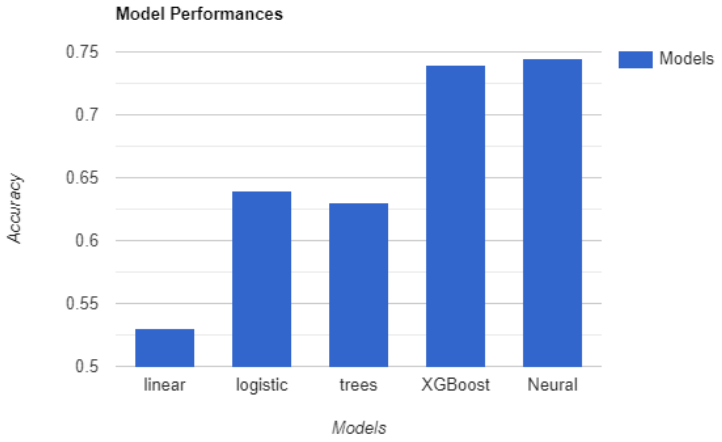
3. Experimental Evaluation

3.1 Methodology

1. We started off simple by trying a linear regression model using sklearn, the model performed very badly as expected and got an accuracy of 0.53.
2. We then tried a logistic regression model, and got a better accuracy with 0.64, which is still not great.
3. Then, we moved to decision trees, and the best model we tried got an accuracy of 0.63.
4. After that, we tried XGBoost and it improved our accuracy massively with an accuracy score of 0.74 which is very close to the targeted accuracy.
5. Lastly, using both Sklearn and Tensorflow, we have trained various neural network, MLP models each one with different hyperparameters and different number of layers and neuron per layers, the best model we could build got an accuracy of 0.745 which is great for this dataset.

3.2 Results

These are the results that we got:



As we can see, both the XGBoost model and the neural network model did great, whereas the other models performed poorly. We decided to continue with the neural network model using tensorflow, and these were the final parameters after tuning:

Nb of layers: 5

Nb of Neurons: 900, 700, 500, 100, 1

Activation: relu, last layer sigmoid

Epochs: 20

Optimizer: adam

Batch Size: 32

Loss function: Custom*

*We have created a loss function that gives a weight of 3 for the last 7 features of the dataset, which are the engineered ones, and a weight of 1 for the remaining 21 input

features, because it proved to be more effective than the standard binary cross entropy loss function. This is our code:

```
def custom_loss(y_true, y_pred):
    weights = tf.constant([1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
    0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 3.0, 3.0 , 3
    .0, 3.0, 3.0, 3.0, 3.0])  # higher weight for the second feature
    weighted_difference = tf.math.multiply(weights, tf.math.subtract(y_tru
    e, y_pred))
    loss = tf.math.reduce_mean(tf.math.square(weighted_difference))
    return loss
```

```
#74.3
model = tf.keras.Sequential([
  tf.keras.layers.Dense(900, activation='relu'),
  tf.keras.layers.Dense(700, activation='relu'),
  tf.keras.layers.Dense(500, activation='relu'),
  tf.keras.layers.Dense(100, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss=custom_loss, metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_cross_val, y_cross_val))

# Evaluate the model on the train, cross validation and test data
train_loss, train_accuracy = model.evaluate(X_train, y_train)
test_loss, test_accuracy = model.evaluate(X_test, y_test)
cv_loss, cv_accuracy = model.evaluate(X_cross_val, y_cross_val)

# Print the test accuracy

print('Train accuracy:', train_accuracy)
print('Cross Val accuracy:', cv_accuracy)
print('Test accuracy:', test_accuracy)
```

3.3 Discussion

We were provided with a table that shows the best possible accuracy for various models, the table shows that for neural networks, a maximum accuracy of 0.78 could be achieved, so having reached 0.745, our model was not far from the best possible, but it could still be updated, however, after trying grid search, randomized search and manually tuning the parameters, we were unable to achieve a better accuracy.

4. Future Work

Our next step is to present the model we have created and justify our choices made along the way that led us to this specific model.

5. Conclusion

Finally, this project was a very interesting and fun, we had to combine all the knowledge that we got from the course and apply that knowledge while testing all the models. This project has been a valuable learning experience in applying machine learning techniques to solve real-world problems.

Github Repository:

https://github.com/MarcMansour02/HIGGS-Boson