SVM and MLP's for Higg's Boson detection

Tyler Liddell

1 Introduction

The Higgs dataset [Whi14] was produced using Monte Carlo simulation and is based on detecting the Higgs boson. This elementary particle is important because its field is responsible for giving mass to the entire universe [CER24]. However, detecting this particle is extremely difficult and is done using a particle accelerator like the Large Hadron Collider (LHC) at CERN. Even though we are now able to produce these particles, detecting them is still a difficult task. The LHC can produce as many as 10^{11} collisions per hour, on average 300 of these will result in a Higgs boson [BSW14]. The Higgs boson has a very short lifetime and decays almost immediately into other particles. This makes direct detection impossible, and scientists rely on identifying the decay products and reconstructing the original particle. [CER24].

Distinguishing Higgs events from the vast background of other particle collisions is a significant challenge. Machine learning techniques, particularly deep learning, have proven to be valuable tools in this endeavor. By training models on simulated data, researchers can develop algorithms capable of identifying Higgs events with high accuracy, even in the presence of substantial noise. The Higgs dataset used in this study is an example of such simulated data, designed to test and refine these machine learning approaches [BSW14]. In this work I compare the SVM and MLP for this task. At the onset I hypothesised that the SVM would be able find linear relationships between the features and would perform closely with the MLP.

2 Methods

2.1 Data analysis

The dataset used is composed of 11 million instances obtained through the usage of a Monte Carlo simulation. Each entry contains a label, and 28 features. The first 21 features are kinematic properties measured by the particle detectors in the accelerator and the last 7 are functions of the first 21 features derived by physicists. The kinematic properties describe the properties of the charged lepton (electron or muon), the missing transverse energy from undetected particles like neutrinos, and up to four reconstructed jets originating from quarks or gluons in a particle collision event. For the charged lepton, the transverse momentum (pT), pseudorapidity, and azimuthal angle are given. The missing transverse energy's magnitude and azimuthal angle are included. For each reconstructed jet, the transverse momentum, pseudorapidity, azimuthal angle, and an indicator is provided for whether each jet originated from a b-quark (known as a b-jet) [BSW14]. Using the random forest algorithm I was able to gain a rough estimate about the performance of each of the features. As seen below in figure 1 the importance of the features is pretty evenly distributed, with a few exceptions. I found however, that removing the less significant features still hurt accuracy enough to warrant keeping them. After the initial training of some basic models, I came back to the data to see

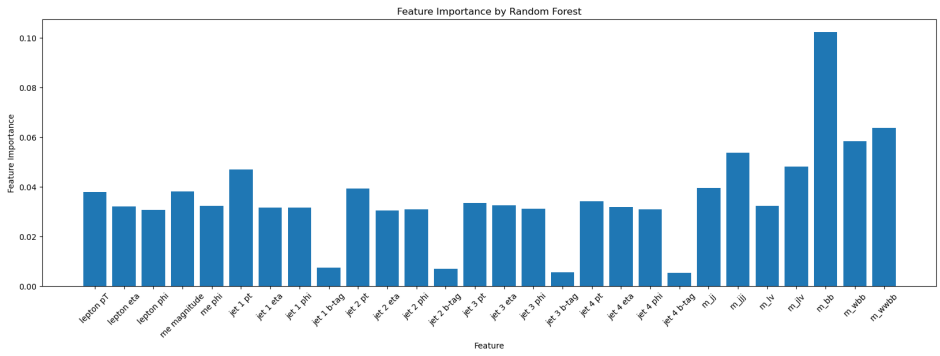

Figure 1: Feature importances obtained using the random forest algorithm.

if further preprocessing could improve performance at all. First I experimented with feature scaling, and found that this sped up, and improved the accuracy of my SVM models. This brings all the units to a similar scale which can improve convergence and reduces the impact of outliers. Another technique I was able to utilize was cyclic encoding. The azimuthal angle is provided for the resulting particles from the collision which describes "the angle between the transverse momentum vector and the horizontal (x) axis[A+17]." In machine learning it is common to capture the cyclic nature of a feature that occurs in a cycle by taking the sin and cos of it. This is most common with units of time, but seeing as an angle is also cyclic I sought to replicate this. There were 6 angular features in the dataset so removing these and adding the sin and cos features resulted in 34 columns instead of 28. I found that these extra features helped the performance of my MLP by improving the accuracy and F1 score by approximately 1 - 2% on average, but had little effect on the SVM. Regardless, I used both of these methods on the training data and for the final test set where I compare an SVM and a MLP.

2.2 Training

2.2.1 Support Vector Machine (SVM)

SVMs are a type of supervised learning algorithm used for classification and regression tasks. The goal of an SVM is to find the optimal hyperplane that maximally separates the different classes in the feature space. By transforming the data into a higher-dimensional space using kernel functions, SVM's can effectively handle non-linearly separable data and create complex decision boundaries.

SVMs can be slow for large datasets, but excel on problems that are linearly separable. In an SVM the main hyperparameters to be explored are the kernel and C (regularization parameter). As stated the kernel is used to transform the data into a higher dimensional space. Popular choices for this are Linear, used to find linear separations; Poly, which finds non-linear decision boundaries based on the degree of the polynomial; RBF, maps data into an infinite-dimensional space, this is effective when there is no prior knowledge about the data; Sigmoid, which is used for binary classification but less common compared to the others [SAA18].

A smaller C value results in a wider margin and allows for more misclassifications, while a larger C value results in a narrower margin and fewer misclassifications. In other words, the C parameter controls the trade-off between allowing misclassifications and forcing strict margins. It determines the balance between achieving a low training error and a low testing

error, thus affecting the bias-variance trade-off of the model. The C value controls how much the model is penalized for misclassifying data points. My goal was to see if the problem could be solved in linear space by training an SVM model on a smaller portion of the dataset. In figure 2 we explore how the different kernels perform starting with a C value of .0001 and doubling each iteration until 1000 using only a few thousand of the training samples. As seen in the graph the linear kernel is able to outperform the others by a noticeable amount, and the best c value occurred at the 26 mark. However, I found that using the linear kernel was not practical if I decided to use more of the training data. When I attempted to train an SVM with the Linear kernel and the optimal c value on a larger portion of the data, the data was no longer linearly separable and the model could not converge. When the amount of training samples were increased the RBF kernel was able to train quickly and its accuracy went up as more training samples were introduced as I will show in the results section. In summary, because the RBF kernel could handle larger amounts of training data it outperformed the linear and other kernels.

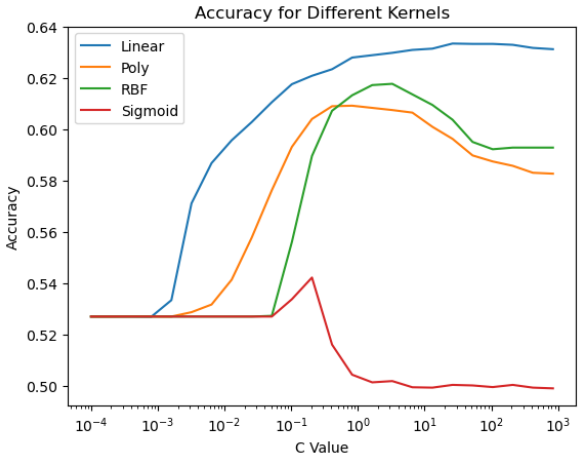

Figure 2: Kernel performances across C.

2.2.2 Multilayer Perceptron (MLP)

MLPs are a type of feedforward artificial neural network consisting of an input layer, one or more hidden layers, and an output layer. Each layer is composed of interconnected nodes (neurons) that process and transmit information forward through the network. MLPs learn to map input features to the desired output by adjusting the weights of the connections between neurons during training, typically using the backpropagation algorithm to minimize the difference between the predicted and actual outputs.

After the first few tests the MLP appeared that it would not achieve much better results than the SVM. There were three parameters that were able to drastically influence the accuracy and F1 score of the model. These parameters were learning rate, activation function choice, and the optimizer used for backpropogation. As can be seen in figure 3 there was a peak in optimum learning rate around .009. This was found using many iterations with a slowly increasing learning rate on the simplest model with 2 hidden layers. I found that hyperparameter tuning done on the smaller models with less training data was effective for improving the larger models trained on more data. Using the Adam optimization function also proved to be the most effective choice. Adam utilizes momentum and adaptive learning rates to accelerate convergence and improve the model's ability to navigate complex loss landscapes. By incorporating exponentially decaying averages of past gradients and squared gradients, Adam dynamically adjusts the learning rate for each parameter based on its historical updates [KB17]. For all of the models tested using either ReLU or Leaky ReLU on each layer proved to be more effective than any of the alternatives. Architecture choices like number of hidden neurons and number of hidden layers had less of an impact on accuracy and f1 score, but greatly impacted training time as I show later.

In an effort to evaluate the different architectures side by side with different learning rates, numbers of hidden neurons, and optimizers, I created 5 different models to be evaluated with

the changes. The first model as discussed had just two hidden layers and the hidden size remained the same in both layers. The second model added a third hidden layer, the hidden size could vary in each layer, and implemented dropout. Dropout is a technique for preventing overfitting, which became an issue as the models became deeper. The key idea of dropout is to temporarily "drop" a fraction of the neurons each forward pass [HSK+12]. The effect of using this technique can be seen in figures 4 and 5 where training and testing losses separate without dropout. Models 3 and 4 had 5 hidden layers and 4 differed from 5 in that it utilized the Kaiming weight initialization. Model 5 had 6 hidden layers, weight initialization, and experimented with batch normalization. The effect of using this technique can be seen in figures 4 and 5 where training and testing losses separate without dropout.

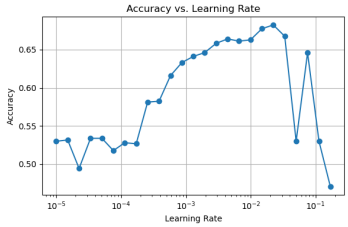

Figure 3: Model accuracy as learning rate goes up

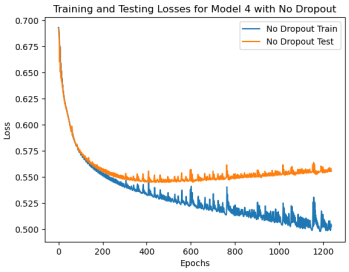

Figure 4: Larger model when trained without dropout

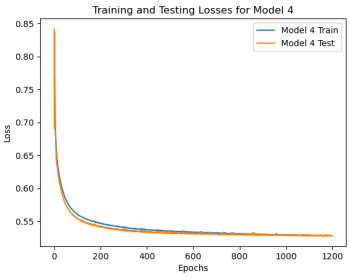

Figure 5: Larger model when trained with dropout

An increase in the number of layers and number of hidden neurons only gave small improvements to the f1 score and accuracy, but greatly increased training time, even when the models were trained on a large GPU. In table 1 we show that as the hidden size and number of layers goes up, training time increases massively while other metrics only increase slightly.

Multilayer Perceptron Performances			
Models	Accuracy	F1 Score	Training Time
Model 1 (2 hidden)	75.0	76.8	7 minutes
Model 2 (3 hidden)	76.3	77.7	9 minutes
Model 3 (5 hidden)	77.2	78.9	21 minutes
Model 4 (5 hidden)	77.1	78.9	26 minutes
Model 5 (6 hidden)	77.6	79.5	38 minutes

Table 1: Comparison of different MLP architectures

Despite this, the architecture chosen for the final test set was Model 5, which consists of 6 hidden layers with decreasing number of neurons in each layer (300, 260, 220, 180, 140, 100, 60). This funnel-like structure allows the model to gradually compress and abstract the input features as they pass through the network. Kaiming weight initialization was used to ensure proper initial weight distribution, and batch normalization was applied after each hidden layer to normalize activations and stabilize training. The ReLU activation function was

selected for its simplicity and effectiveness in introducing non-linearity. Additionally, dropout with a probability of 0.2 was applied after each hidden layer to regularize the model and prevent overfitting.

3 Results

When comparing both the SVM and the MLP for this task, the only aspect where the SVM has an edge is training time, but this is distorted due to the SVM needing to be trained on less data. The SVM was trained in two and a half minutes using only a CPU, while the MLP for this comparison (Model 5) took 38 minutes to train on a GPU. This advantage was quickly diminished when also taking into account prediction time. When the models were tested on 50,000 samples of the test data, the SVM took 3 minutes 14 seconds, while the MLP was able to make predictions in just under half of a second. The confusion matrices presented in Figures 6 and 7 reveal that both models exhibit a relatively balanced performance across the different classes. However, there is a slight bias towards predicting the signal class more frequently compared to predicting background.

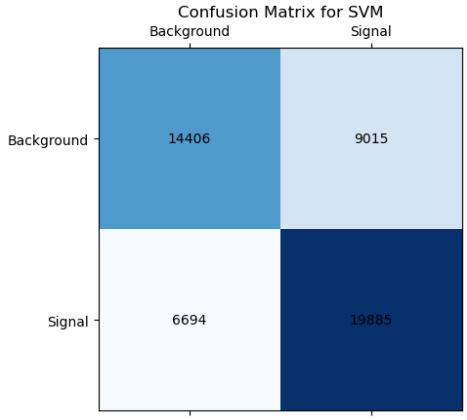

Figure 6: Confusion matrix for SVM

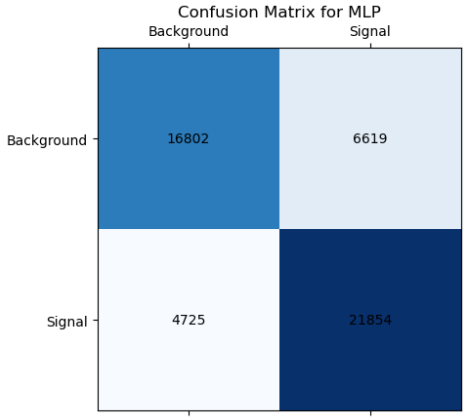

Figure 7: Confusion matrix for MLP

Overall the MLP performed much better across all metrics apart from training time. On the final test set the MLP had an accuracy of 77.6% and an F1 score of 79.4%, while the SVM had an accuracy of 68.7% and an F1 score of 71.7%. This significant difference in performance can be attributed to the MLP's ability to learn complex, non-linear relationships between the input features and the target variables.

4 Conclusion

In conclusion, this study demonstrates the effectiveness of machine learning techniques, particularly Support Vector Machines (SVM) and Multilayer Perceptrons (MLP), in detecting Higgs boson events from a large background of particle collisions. Through careful preprocessing of the data, including feature scaling and cyclic encoding of angular features, the performance of both models was optimized.

The SVM, while faster to train, ultimately fell short in terms of accuracy and F1 score compared to the MLP. In light of my hypothesis, the SVM proved to be insufficient for this task and was unable to find a linear relationship between the features. However, the MLP's ability to

learn complex, non-linear relationships proved crucial in this challenging classification task. The final MLP architecture, consisting of 6 hidden layers with decreasing neuron counts, regularization techniques like dropout and batch normalization, and the ReLU activation function, achieved an impressive 77.6% accuracy and 79.4% F1 score on the test set.

Despite the MLP's longer training time, its rapid prediction speed on new data makes it a promising tool for real-time event classification in high-energy physics experiments. The results underscore the potential of deep learning in aiding physicists to sift through the vast amounts of data generated by particle accelerators and identify rare, significant events like Higgs boson decays.

Future work could explore more advanced neural network architectures, such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), to potentially capture additional spatial or temporal patterns in the data. Other idea could be ensembling multiple models or incorporating domain knowledge through physics-informed architectures to further improve classification performance.

Overall, this research highlights the synergistic relationship between machine learning and particle physics and demonstrates how AI can accelerate scientific discovery in the search for fundamental particles.

References

[A+17] A. Aduszkiewicz et al. Two-particle correlations in azimuthal angle and pseudorapidity in inelastic p + p interactions at the cern super proton synchrotron. *The European Physical Journal C*, 77(2), Jan 2017.

[BSW14] P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5(1), July 2014.

[CER24] CERN. What's so special about the Higgs boson?, Accessed 2024. Accessed on February 26, 2024.

[HSK+12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[SAA18] Intisar Shadeed, Dhafar Abd, and Jwan Alwan. Performance evaluation of kernels in support vector machine. pages 96–101, 11 2018.

[Whi14] Daniel Whiteson. HIGGS. UCI Machine Learning Repository, 2014. DOI: https://doi.org/10.24432/C5V312.

5 Glossary

Higgs boson: An elementary particle whose field is responsible for giving mass to the entire universe.

Large Hadron Collider (LHC): A particle accelerator at CERN used to produce and detect particles like the Higgs boson.

Monte Carlo simulation: A computational technique that uses random sampling to model and analyze complex systems, such as particle collisions.

Kinematic properties: Physical quantities that describe the motion of particles, such as transverse momentum, pseudorapidity, and azimuthal angle.

Transverse momentum (pT): The component of a particle's momentum that is perpendicular to the beam axis.

Pseudorapidity: A spatial coordinate that describes the angle of a particle relative to the beam axis.

Azimuthal angle: The angle between the transverse momentum vector and the horizontal (x) axis.

Feature scaling: A preprocessing technique that transforms features to have zero mean and unit variance, bringing them to a similar scale.

Cyclic encoding: A technique that captures the cyclic nature of a feature by taking the sine and cosine of its values.

Support Vector Machine (SVM): A supervised learning algorithm used for classification and regression tasks, which aims to find the optimal hyperplane that separates different classes.

Kernel functions: Mathematical functions used in SVMs to transform data into a higher-dimensional space, enabling the creation of complex decision boundaries.

Regularization parameter (C): A hyperparameter in SVMs that controls the trade-off between allowing misclassifications and forcing strict margins.

Multilayer Perceptron (MLP): A type of feedforward artificial neural network consisting of an input layer, one or more hidden layers, and an output layer.

Backpropagation: An algorithm used to train MLPs by adjusting the weights of the connections between neurons to minimize the difference between predicted and actual outputs.

Activation function: A mathematical function applied to the weighted sum of inputs in a neural network to introduce non-linearity and determine the output of a neuron.

Dropout: A regularization technique used in neural networks to prevent overfitting by temporarily "dropping" a fraction of the neurons during each forward pass.

Kaiming weight initialization: A method for initializing the weights of a neural network to ensure proper initial weight distribution.

Batch normalization: A technique used in neural networks to normalize activations and stabilize training by adjusting the mean and variance of the activations within each batch.

Confusion matrix: A table that summarizes the performance of a classification model by showing the counts of true positives, true negatives, false positives, and false negatives.

F1 score: A metric that combines precision and recall to provide a single score that balances both metrics, often used to evaluate the performance of a classification model.