

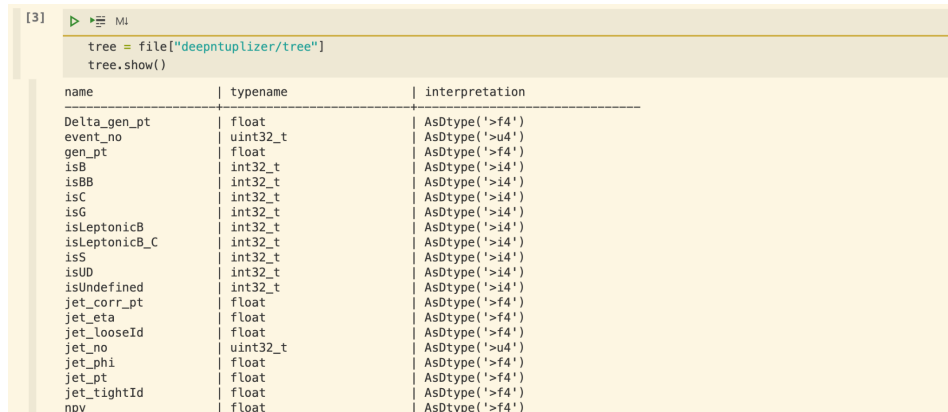
Neural Network for Higgs Detection

Noah Zipper and Alec Emser
PHYS 5070 - Spring 2021

Training Data and Physics Background

Training Data

We pull our training and test datasets from CERN's open data portal¹. The dataset is made up of particle jets from simulated proton-proton collisions with a Center-of-Mass energy of 13 TeV, and was created for the express purpose of training machine-learning algorithms to differentiate between jets arising from quantum chromodynamic production and jets arising from a Higgs to bottom quark-antiquark pair decay. The process of pulling the data set and selecting features occurs in “buildDataSets.py”. Feature selection is described in greater detail below.



```
[3] In [ ]: tree = file["deepntuplizer/tree"]
tree.show()
```

name	typename	interpretation
Delta_gen_pt	float	AsDtype('>f4')
event_no	uint32_t	AsDtype('>u4')
gen_pt	float	AsDtype('>f4')
isB	int32_t	AsDtype('>i4')
isBB	int32_t	AsDtype('>i4')
isC	int32_t	AsDtype('>i4')
isG	int32_t	AsDtype('>i4')
isLeptonicB	int32_t	AsDtype('>i4')
isLeptonicB_C	int32_t	AsDtype('>i4')
isS	int32_t	AsDtype('>i4')
isUD	int32_t	AsDtype('>i4')
isUndefined	int32_t	AsDtype('>i4')
jet_corr_pt	float	AsDtype('>f4')
jet_eta	float	AsDtype('>f4')
jet_looseId	float	AsDtype('>f4')
jet_no	uint32_t	AsDtype('>u4')
jet_phi	float	AsDtype('>f4')
jet_pt	float	AsDtype('>f4')
jet_tightId	float	AsDtype('>f4')
npv	float	AsDtype('>f4')

We use the Uproot library to work with this data without the explicit use of ROOT, the esoteric analysis framework typical (but increasingly less so) for particle physics research. In conjunction with the XRootD library, which allows us to access the remote repositories where the training and test data is stored without needing to download files (O(100 GB)), we can directly read our datasets into Pandas Dataframes for easy manipulation and formatting.

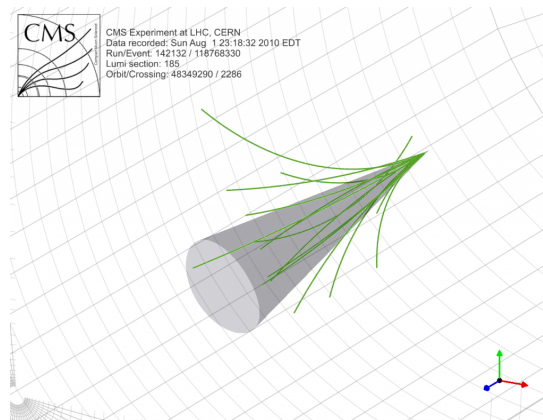
We use our feature selection studies to choose an appropriate set for classification, as well as the true labels, and generate a substantial number of both training and testing examples to be used with our model. Keeping the training and testing sets in separate files allows all of our training and optimization to occur without ever accessing the test set, so as to avoid implicit biases.

¹ <http://opendata.cern.ch/record/12102#>

Physics Background

In order to build an effective model for classifying Higgs Bosons, we generate an optimal set of features for our neural network to use. The first step is understanding the physical motivation behind each feature, so as to not treat this framework as a black box.

The physics object of interest in this classification task is called a “jet”. This term refers to a shower, or a geometrical cone, of particles that are typically detected as a clustered energy deposit in the electromagnetic and hadronic calorimeter of the CMS detector, as a product of proton-proton collisions. Furthermore, charged jets can be reconstructed to produce one or more “tracks”, or a calculated trajectory inside the inner tracker of the detector. These jets come in multiple flavors, which correspond to the algorithm used to cluster its energy deposits. In our case we have AK8 jets (also called fat jets), which use the Anti- K_T algorithm and a cone size of 0.8, and the skinnier AK4 jets. The important thing to consider here is that a heavier particle like the Higgs, which can decay into two separate quarks, will all be clustered together into a larger object, meaning we identify them as AK8 jets. Along the same lines, the smaller clusters that correspond to the individual b-quarks, or other light quarks we may find in the calorimeter, correspond to AK4 jets.



<https://www.quantumdiaries.org/tag/jets/>

Each element of these datasets correspond to a given jet in a simulated CMS event. These events are derived from two separate Monte Carlo sources: Gravitons that decay to two Higgs Bosons, each of which decay to $b\bar{b}$ pairs, and QCD background. This QCD source, without getting too specific, represents noise that doesn't correspond to the consistent production of any other particle of interest, produced uniformly across the momentum spectrum. All samples are produced as simulated production for the Run II Summer 2016 timeframe, so no additional reweighting is required. Much more can be said about these samples and the nuances of the physics behind the two processes we hope to differentiate, but none of this holds a significant bearing on the performance or design of our estimator, so we will leave it here.

To briefly summarize the immense list of TBranches (ROOT jargon that mostly correspond to features) available in the OpenData files, we have:

- Number of primary vertices in the jets event
- A number of truth-level labels (the true jet identity jet that we know because we simulated them)
- Typical kinematic quantities: transverse momentum (p_T), η (polar angle relative to the beam line), and ϕ (azimuthal angle)
- Tau variables, or constructed predictive metrics that use clustering information to estimate the number of sub-jets contained within a candidate jet
- Geometrical information on all the reconstructed tracks corresponding to a candidate jet
- Variables corresponding to “Particle-Flow Candidates”, high-level physics objects obtained by combining information from different detector systems (the tracker, calorimeters, and sometimes muon system)

This is a very rough paraphrasing of all the information that is given to us, but for the purposes of model construction, it gives us an idea of what components from a given proton-proton collision are used as information to identify our object of interest. In a typical physics analysis, a classifier such as ours is used (referred to as a tagger), along with its corresponding uncertainties, to identify particles in both the MC simulated signal events and data. These expected yields and their agreement give us an estimate as to whether or not, with our current experimental sensitivity, anything of interest is found.

The newest version of the official CMS analog to our Hbb tagger (DeepDoubleB), is currently being used in an ongoing Supersymmetry (SUSY) analysis by the CU CMS group, where it is used to find evidence for Higgsinos and Gluinos. These supersymmetric particles decay in what we call a “Boosted HH” topology, where they produce two high-energy Higgs Bosons that, because of their “boosted” momentum, can each be clustered into single AK8 jets. The performance of the DeepDoubleB tagger is critical to the sensitivity of this analysis.

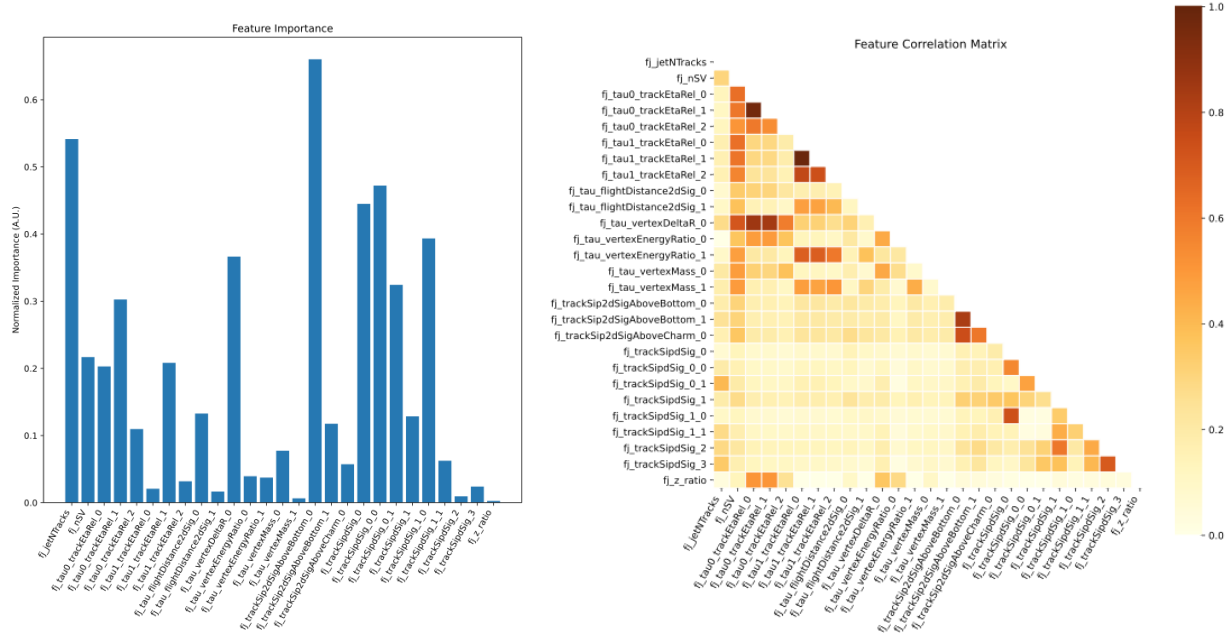
Feature Selection and Model Building

Feature selection

Taking the information from our physics background, and relying on the general list of variables relevant to heavy-flavor tagging presented by CMS², we generated an initial list of >30 features that seemed relevant. We perform feature studies in the “featureModelling.ipynb” Jupyter Notebook. To gauge the impact that each feature has individually, we constructed a basic logistic regression classification model with the Keras API, and fit it to our data set. By normalizing all the features, we can then use the coefficients associated with each feature for the linear classifier to understand the impact that each feature has on the model’s prediction. For the final set of features chosen, the feature importance is shown below.

² Sirunyan, A., Tumasyan, A., Adam, W., Ambrogio, F., Asilar, E., Bergauer, T., Brandstetter, J., Brondolin, E., Dragicevic, M., Erö, J., & et al. (2018). Identification of heavy-flavour jets with the CMS detector in pp collisions at 13 TeV. *Journal of Instrumentation*, 13(05), P05011–P05011.

Similarly, we check for correlations between different features in our set. This can be done directly in Pandas, and gives us an idea of what redundancies we may have in our feature set. If multiple features are highly correlated, their discriminative impact may be redundant, and we risk overcomplicating our model with high-dimensionality. The correlation matrix for the final feature set is also presented below.



The features shown in the figures above correspond to the final set. This was in accordance with the suggestions by CMS, but also pared down by our studies. While there may be instances of features that seem unimportant or highly correlated in this collection, we made thoughtful judgement calls based on our understanding of the quantities themselves, or based on the recommendations from CMS. For instance, “`fj_tau1_trackEtaRel_1`” and “`fj_tau1_trackEtaRel_2`” seem to be highly correlated, and the first one is much more important according to our logistic regression test. This is because they are both track angles relative to the tau 1 “N-subjettiness axis”. While it may seem that this means that we can drop the second feature, because these correspond to different tracks, the second variable may become necessary in cases where the closest track can be discounted for other kinematic reasons such as low p_T . In these cases, using the second track may allow for better discrimination. So while it may not be useful in 99% of events, and thus score low on our feature importance test, it can offer additional sensitivity to particular edge cases.

Model Architecture

Once our feature set was selected, we focused on optimizing our model architecture. This was done using the GridSearchCV method in SciKit Learn, which performs an exhaustive grid-search over a selection of model hyperparameters, and using a cross-validated accuracy metric, selects the combination of hyperparameters to give us the best performance. The model training happens in the “trainNNModel.py” Python script. The following hyperparameters, the options we suggested, and the optimal value are as follows:

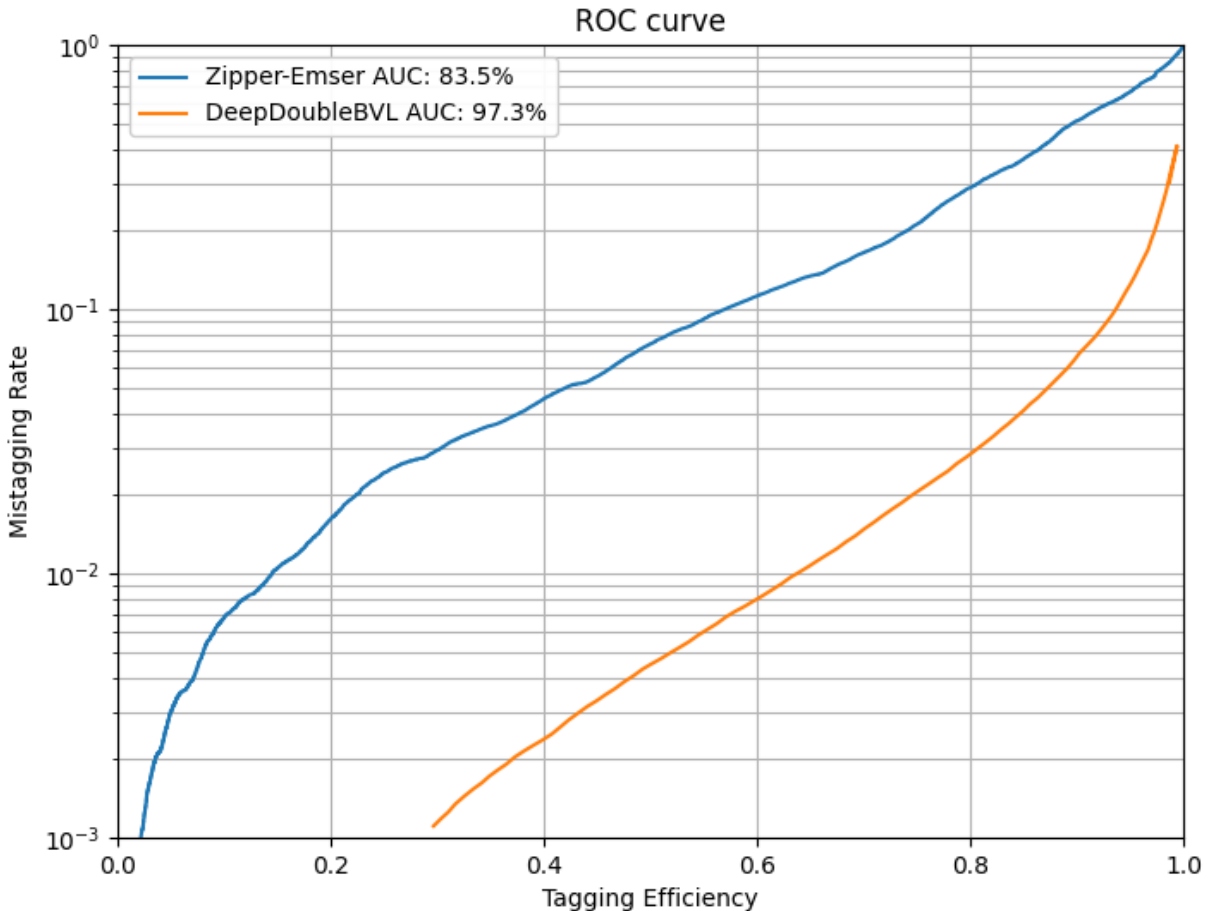
- Training epochs: # of times to run through entire training set
 - [2, 5] → 2
- Training batch size: # of examples to pass through network before 'self-correcting'
 - [2, 5] → 5
- Number of hidden layers in neural network:
 - 3
- Hidden layer activation function:
 - Relu (rectified linear unit)
- Hidden layer dropout: % of nodes to omit in hidden layers in order to regularize output
 - [0.2, 0.5] → 0.2
- Hidden layer width:
 - 25
- Optimizer: algorithm to carry out stochastic gradient descent
 - adam

Performance Analysis

We use the Jupyter notebook “testNNmodel.ipynb” to benchmark the performance of our model. We can test the absolute accuracy of the neural network on both the dataset used to train the model and on the partitioned testing dataset. We find that the model has an approximately 87.8% accuracy on both the training set and test data set. The roughly equal accuracy between sets suggests that our algorithm is not overfitting to the training data.

Training Accuracy	87.81%
Test Accuracy	87.78%

Of great concern is of course the false positive rate (or mistagging rate) in comparison to the true positive rate (the tagging efficiency). It would be of little help to have an algorithm which captures every Higgs event while labeling many other events also as Higgs events. Below we have produced a receiver operating characteristic curve (ROC curve) which characterizes this behavior by plotting the mistagging rate against the tagging efficiency. A crucial metric for ROC curves is the area under curve (AUC), a measure of aggregate performance by integrating the area under the curve of tagging efficiency against mistagging rate. The AUC is both scale-invariant and classification threshold invariant, making it an ideal measure for comparing competing algorithms. It can be thought of as the specific probability that the algorithm will score a true positive event (Higgs decay) with a greater likelihood than a negative event (not Higgs decay). Our own algorithm, plotted in blue, scores an AUC of 83.5%.



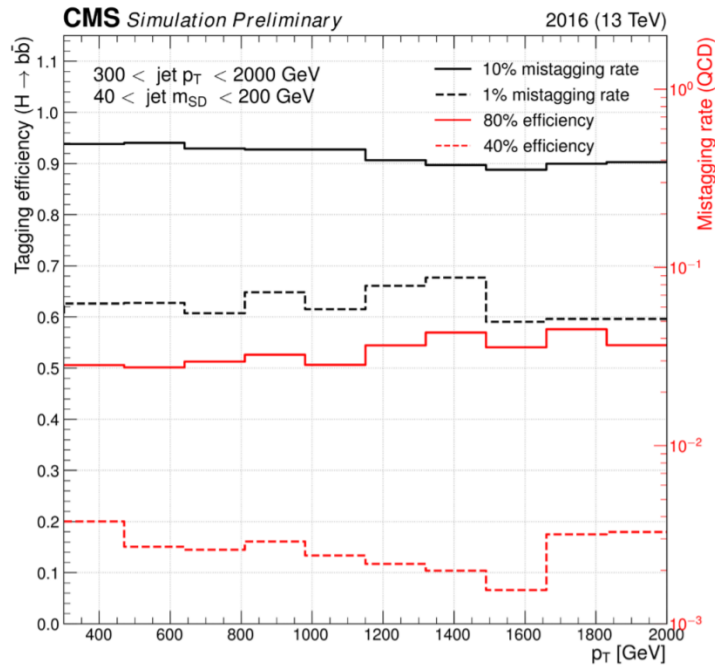
We can compare our algorithm with the [DeepDoubleBvL](#) algorithm, CMS' own 2018 neural network used for identifying Higgses to bottom quark-antiquark decays. Their algorithm is based on the same 27 features as our own, but includes additional sampling of jet constituent properties which are trained with their own convolutional layers and fed into gated recurrence units (GRUs) before being merged with the original 27 jet features and fed into one final dense convolutional layer with 100 nodes. The DeepDoubleBvL performance is plotted next to our algorithm above.

We see that our algorithm has a higher mistagging rate than the DeepDoubleBvL algorithm for all tagging efficiencies, leading also to an AUC $\sim 14\%$ lower. This is most likely a result of the additional properties of the charged particle-flow jet constituents which the algorithm takes into account. The original [double-b tagger](#)¹ scored an AUC of 91.3%, significantly closer to our own performance. While our algorithm scores below these official CMS publications, we consider the performance to be fairly good for a course final project.

Future Directions

There are many ways in which we could imagine continuing this project. One initial goal was to test our model with respect to important kinematic variable, to see how our identification

accuracy changes with p_T or its mass (soft-drop mass for the case of jets). These corresponding results are shown for the DeepDoubleB tagger below.



In addition to this result, there are numerous ways we could improve our model. By only using our laptops, exhaustive grid-search, even when utilizing multiple cores in parallel, is incredibly time-consuming. By using a more powerful computing grid, we could have further parallelized this process and added additional dimensionality to our grid-search, finding better hyperparameters. Finally, attempting the convolutional architecture of the DeepDoubleB neural network would most likely get us closer to matching their accuracy, but being able to regularize and apply filters necessary for convolutions is much more difficult to do with the format of the toy datasets we used in our project.

In any case, I believe we provided a simple and successful template for tagging Higgs Bosons in CMS events, using basic data analysis tools and techniques.