TRT Machine Learning Particle ID

Doug Davis, Matthew Epland, Sourav Sen

Duke University

Feb 2018

Introduction

- R&D study of e-μ particle ID (PID) in the ATLAS TRT sub-detector using machine learning techniques
- Try to improve on the existing eProbabilityHT likelihood function
- Tested Support Vector Machines (SVM), Boosted Decision Trees (BDT), and Neural Networks (NN)
  - Continuation of Doug's SVM work last semester
- Supervised learning using Monte Carlo (MC) truth PID
- Implemented with scikit-learn and Keras+TensorFlow

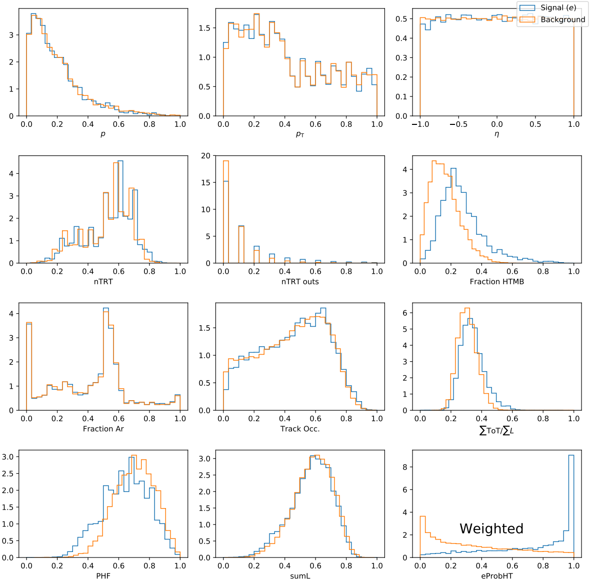
Data Preparation

- MC16 $Z \rightarrow ll$ events
- e & μ selections:
  - Lepton and track p and $p_{\mathrm{T}} > 5$ GeV
  - ≥ 1 pixel hit
  - ≥ 6 silicon hits
  - ≥ 12 TRT hits
  - Truth matched to a Z decay
- Also require ≥ 1 precision hit for muons
- Results in a training set of $m =$ 1.4M leptons
  - Testing on 0.29M leptons, 20% of total
  - Test set split 50-50 between e & μ
- Reweight events to have a flat p_{T}, η distribution
  - Try to avoid biasing results based on energy or location in detector, while still allowing p_{T} and η to be used

Input Variables / Features

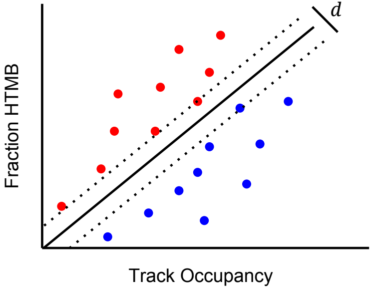
- Primarily using track based variables:
    - p, p_{T}, η
    - nTRT: Number of TRT hits from TrackSummary
        - Precision + tube, no outliers
    - nTRT outliers: Number of outlier hits (outside of straw)
    - Fraction HTMB: Fraction of high threshold hits
    - Fraction Ar: Fraction of Ar hits
    - PHF: Fraction of precision hits
    - Track Occupancy
    - $\sum L$: $\sum$ all track L in straws
    - $\sum \mathrm{ToT} / \sum L$: $\sum$ all time over thresholds / $\sum$ all track L in straws
- Also use existing eProbabilityHT, for $n = 12$ variables in total
- Scale each to $[0, 1]$ to help convergence (particularly for SVM)
    - Scale based on training data, apply to training and testing
    - Keep η symmetric $[-1, 1]$

Input Variables



Support Vector Machines

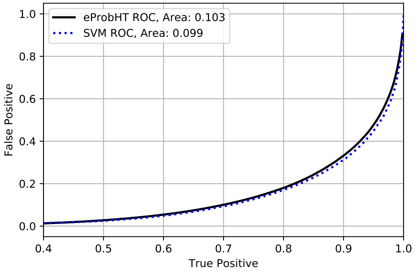
- Draw hyper-plane in n dimensional space to separate classes
- Optimize margin d between the classes

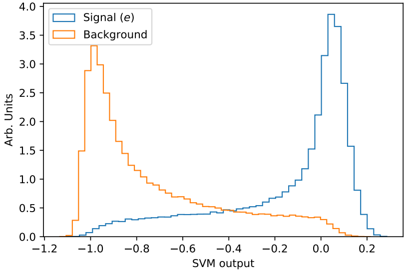


- See here and here for more

SVM Results

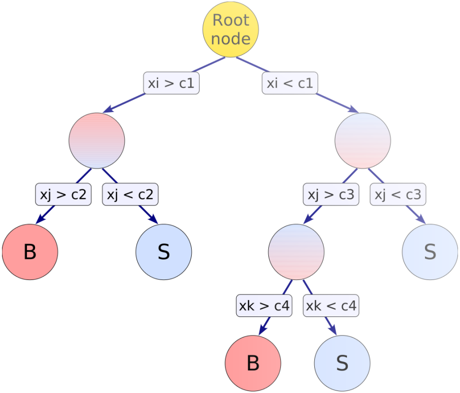
- Use default configuration from sklearn
- Equivalent performance to eProbabilityHT alone
  - For this ROC curve the lower right corner / smaller area is better
- Limit number of training events to $m = 50\text{k}$
  - For practicality as SVM training time goes like $\sim n\, m \log m$





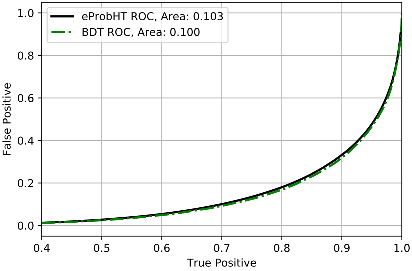
Boosted Decision Trees

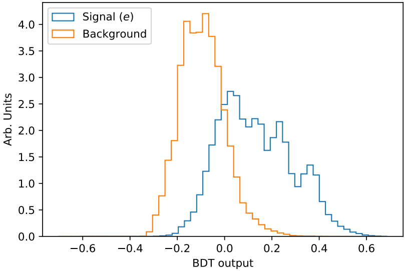
- Start with a simple decision tree, i.e. ordered set of cuts on features
  - Split values are chosen to maximize S and B separation
- Assign each event to a leaf with weight $w_j < 0$ for B, > 0 for S
- Iterativly add additional trees to complement earlier trees (boosting)
- Sum the individual w_j an event recieves from each tree
- See here, here, and here for more



BDT Results

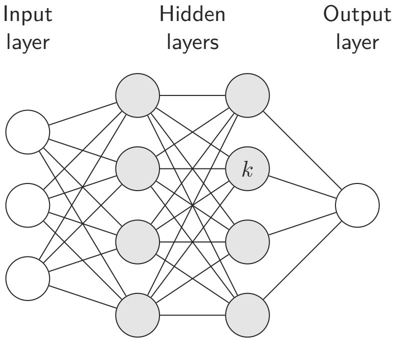
- Use sklearn's AdaBoostClassifier
- Equivalent performance to eProbabilityHT alone

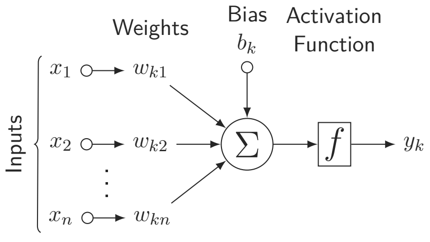




Neural Networks

- Network of connected neurons arranged in layers
- Each neuron defined by weights and a non-linear activation function
- Train network over numerous example to classify high-dimensional data
- Use a flavor of gradient descent (Adam) to find weights by optimizing the loss function (binary cross-entropy) over many iterations (epochs)



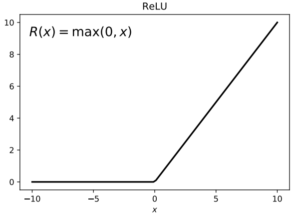


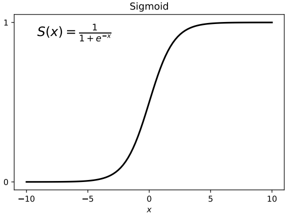
$$y_k = f\left(b_k + \sum_{i=1}^{n} w_{ki}\, x_i\right)$$

NN Setup

- Default: 2 layer network of 12 & 8 nodes, 1 output layer
    - Also tried wider and deeper networks, had very similar performance
- Use ReLU activation function and sigmoid to get [0, 1] output

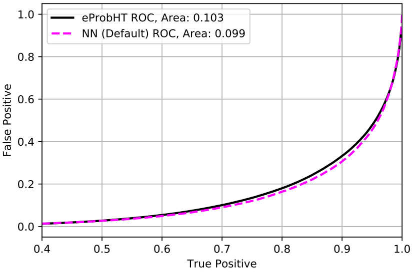
```
model_default = Sequential()
model_default.add(Dense(12, input_dim=input_ndimensions, activation='relu'))
model_default.add(Dense(8, activation='relu'))
model_default.add(Dense(1, activation='sigmoid'))

model_default.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

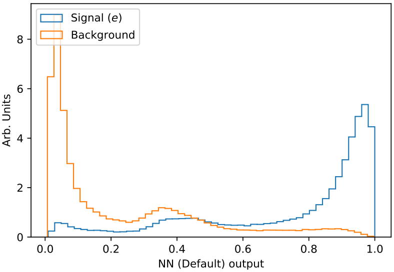




NN Results

- Slightly better performance than eProbabilityHT, SVM, and BDT
- Interesting feature around output scores of 0.35



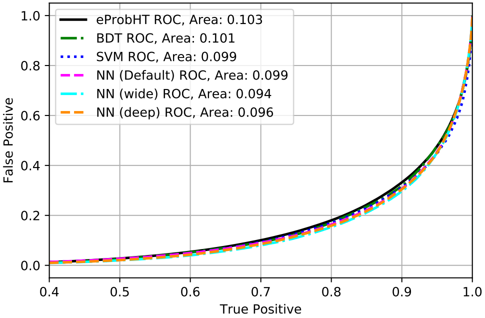


Wider and Deeper Networks

```
wide = Sequential()
wide.add(Dense(24,
        input_dim=input_ndimensions,
        activation='relu'))
wide.add(Dense(16, activation='relu'))
wide.add(Dense(1, activation='sigmoid'))
```

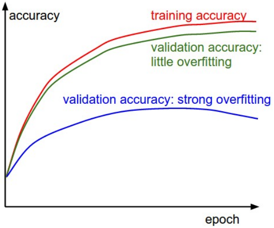
```
deep = Sequential()
deep.add(Dense(12,
        input_dim=input_ndimensions,
        activation='relu'))
deep.add(Dense(8, activation='relu'))
deep.add(Dense(8, activation='relu'))
deep.add(Dense(8, activation='relu'))
deep.add(Dense(8, activation='relu'))
deep.add(Dense(1, activation='sigmoid'))
```

- Try doubling layer width, and adding additional fully connected layers
- No notable differences from the default network



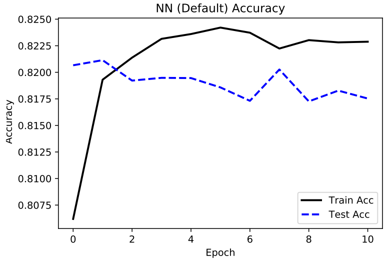
NN Convergence

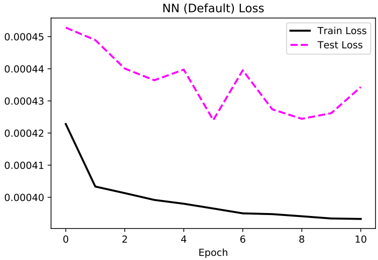
- Want to avoid under and overfitting to our particular training data



Overfit Accuracy (Source)

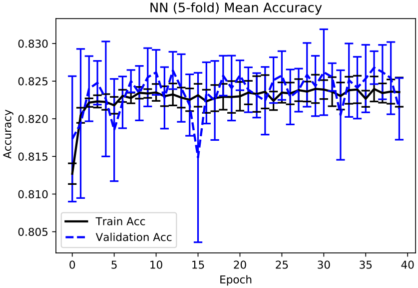
- Are slightly overfitting, but this can be fixed with regularization

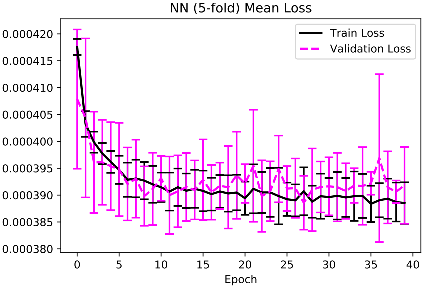




k-fold Cross-Validation

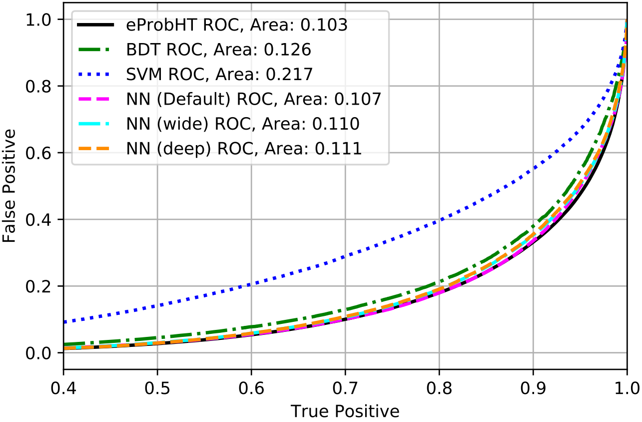
- Goal: Generalize away choice of training and validation data
- Randomly partition training set into $k = 5$ subsets
    - Stratified: Keep e/μ ratio similar in each subset
- Train network k times, using each subset for validation once
- Average accuracy and loss results – Doesn't appear as overfit





Results without eProbabilityHT

- Training without using eProbabilityHT, just the $n = 11$ track variables
- NN can recreate eProbabilityHT's performance, SVM & BDT fall short



Summary

- No results at this stage substantially surpassed eProbabilityHT
- Without using eProbabilityHT, NN can learn enough to match it

- Future Work:
  - Try adding dropout layer or L2 regularization to address slight overfitting
  - Try other sets of input variables
  - Try deeper BDTs
  - Try other sets of input variables
  - Try training with tagged data
  - Mix J/ψ events into dataset
  - Try a recurrent neural network (RNN) with long short-term memory (LSTM) layers to utilize the underlying hits information, including raw bitstream

- Code can be found at github.com/dukeatlas/trtmachana