Classification Methods for t^* Search

Undergraduate Research Report

Nico Braukman · April 2022

Abstract

A summary of the multivariate analysis (MVA) methods used for classification of events in the search for $t^*\bar{t}^*$ pair production in the CMS experiment in presented. Signal events are the pair production of excited top quarks t^* (sometimes denoted T) which decay into the lepton + photon + jets final states: $t^*\bar{t}^* \to t\bar{t}\gamma g \to l{+}\gamma{+}$jets. Several MVA methods are investigated here: an Adaboost boosted decision tree (BDTA) with the TMVA package, a BDT with the XGBoost package, and five neural networks (NN) with the Keras package. The discriminant distributions and receiver operating characteristic (ROC) curves are computed for each method. The upper limits on the signal strength (signal cross section) at 95% confidence level are also computed for the BDTA (TMVA) and NN (Keras) models.

I Background

The exited top quark t^* is a particle predicted to exist in some beyond the standard model (BSM) frameworks which consider the top quark t as a composite rather than an elementary particle[1]. Experimental discovery of the t^* would provide direct evidence for the composite nature of the top quark. The High Energy Physics research group at Florida Institute of Technology is performing a search for excited top quark events ($t^*\bar{t}^* \to t\bar{t}\gamma g \to l + \gamma + \text{jets}$) in data from the CMS experiment at CERN's Large Hadron Collider. This document focuses on events from the 2016 CMS data in the muon channel.

The Feynman diagram for the signal-region decay of the $t^*\bar{t}^*$ pair is shown in Figure 1. Here, one top quark decays leptonically (producing the final-state lepton l, a muon in this analysis), and the other top quark decays hadronically (producing three jets from b, q', and $\bar{q}$). In the "resolved" region, the three jets are well-separated in space; in the "boosted" region, they overlap.

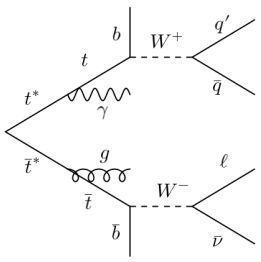


Figure 1: Feynman diagram for $t^*\bar{t}^*$ decay into $t\gamma tg$, where the tops decay semileptonically.

Several variables could in principle be used as observables to separate these signal events from the background. A multivarate analysis (MVA) is performed to account for all of these variables. Table 1 lists all variables used in the resolved and boosted regions. There were 15 (24) total variables in the resolved (boosted) region[3], where the boosted region includes extra variables for the "fat jet" combination of the three unresolved jets from the hadronically-decaying top quark. Variables prefixed "`Reco_`" are reconstructed.

Table 1: MVA Variables

Description	Variable
Resolved and Boosted Regions	
Mass of t^*	`Reco_mass_T`
Mass of $\mu\gamma$	`Reco_mass_lgamma[0]`
η of leading μ	`Muon_eta[0]`
ϕ of leading μ	`Muon_phi[0]`
Number of jets	`Jet_size`
η of leading jet	`Jet_eta[0]`
ϕ of leading jet	`Jet_phi[0]`
Missing transverse energy E_T^{miss}	`Reco_met`
η of leading photon	`Photon_eta[0]`
ϕ of leading photon	`Photon_phi[0]`
Sum of p_T for all objects	`Reco_st`
Angle between leading jet and E_T^{miss}	`Reco_angle_leadJet_met`
Angle between leading γ and μ	`Reco_angle_leadPhoton_lepton`
Angle between lepton (μ) and E_T^{miss}	`Reco_angle_lepton_met`
b-tag discriminator	`Jet_deep_b[0]`
Boosted Region Only	
p_T	`FatJet_pt[0]`
η	`FatJet_eta[0]`
ϕ	`FatJet_phi[0]`
Mass	`FatJet_mass[0]`
Softdrop corrected mass	`FatJet_msoftdrop[0]`
DeepCSV	`FatJet_btagDeepB[0]`
DeepBoostedJet top vs QCD (mass decorr.)	`FatJet_deepTagMD_TvsQCD[0]`
DeepBoostedJet top vs QCD	`FatJet_deepTag_TvsQCD[0]`
DeepBoostedJet W vs QCD	`FatJet_deepTag_WvsQCD[0]`

Thorough descriptions of the datasets, high-level triggers, and event selection cuts used in the analysis will be given in the Analysis Note for this project, and they are not covered here.

Figures 2 and 3 show the correlation matrices of the variables analyzed in the resolved and boosted regions, respectively. In the resolved region, the most highly-correlated variables are `Reco_mass_T` and `Reco_st`, with a correlation coefficient of +71%, which agrees with the expectation that a high-mass t^* will be correlated with high total p_T for the event.

In the boosted region, the `FatJet_mass[0]` and `FatJet_msoftdrop[0]` variables are highly correlated (+89%), as expected, since both are measurements of the mass of the fat jet. Similar correlations are seen in the boosted and resolved regions for the shared variables. Most variables used in the MVA are neither significantly correlated nor anti-correlated with any other variable, as desired. Figures 4 and 5 show the distributions for each variable in the boosted and resolved regions, respectively.

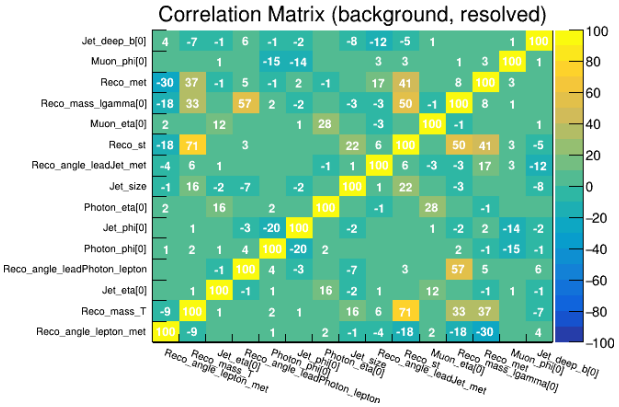


Figure 2: Correlation matrix, resolved region.

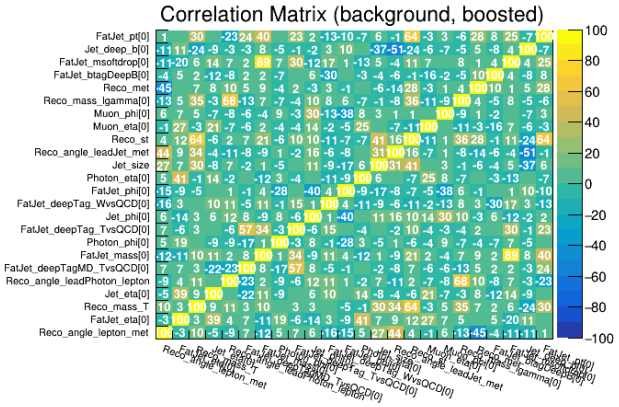


Figure 3: Correlation matrix, boosted region.

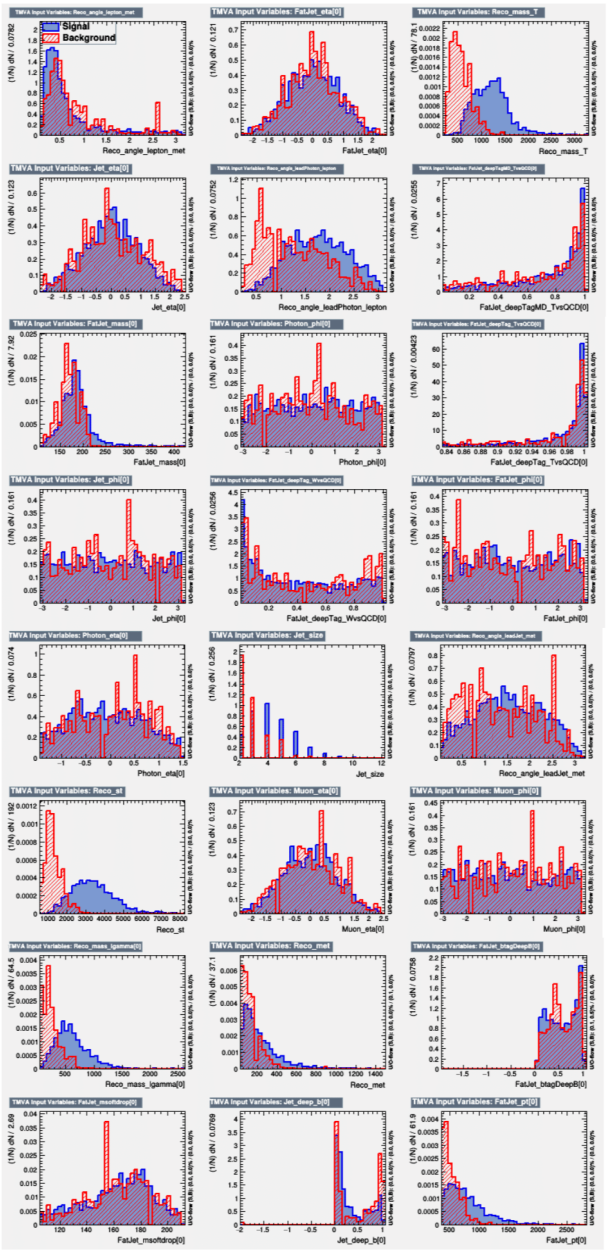


Figure 4: Variable distributions, boosted region.

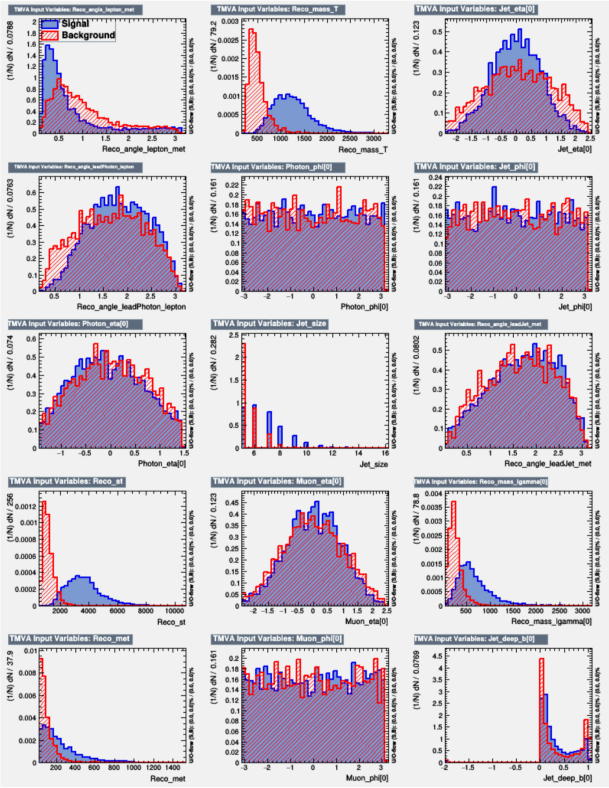


Figure 5: Variable distributions, resolved region.

The following sections describe in detail the MVA methods investigated. Section II describes the BDTA model created within ROOT's Toolkit for Multivariate Analysis (TMVA), section III describes the BDT model created with XGBoost, and section IV describes the neural network models created with Keras. Finally, in section V, the 95% CL upper limits are computed and compared for the BDTA (TMVA) and NN (Keras) models.

II TMVA BDTA MODEL

The BDTA model described here is the same as in [3], and it is the model ultimately used and described in the Analysis Note. It is included in this document for comparison with the XGBoost BDT and Keras NN models, which are not described in the AN.

The BDTA model was constructed using the Toolkit for Multivariate Analysis (TMVA) [2]. First, the signal and background datasets are added to a DataLoader object, as shown in the code below, where "`sigTree`" and "`bkgTree`" are previously loaded TTrees containing the signal and background event data. A directory for the output ROOT file is passed as an argument to the loader.

```
loader = ROOT.TMVA.DataLoader(outDir)
sigWeight = 1.0
bkgWeight = 1.0
loader.AddSignalTree(sigTree, sigWeight)
loader.AddBackgroundTree(bkgTree, bkgWeight)
```

Next, the MVA variables are added to the loader, and the weight expressions are set. Here, "`varDict`" is a dictionary containing the variables in Table 1. The event cuts are applied to the loader using a TCut object, and the events which pass the cut are separated into training and testing sets using the `PrepareTrainingAndTestTree()` method, which selects half of the events at random for training and the other half for testing. These steps are shown in the code below.

```
# Add variables
for var in varDict.keys():
    loader.AddVariable(varDict[var][0], 'F')

# Set weight expressions
evtWeights_sig = "Weight_pu*Weight_mu* ...
        Weight_ele*Weight_prefire* ...
        Weight_btag*Weight_pho[0]"
evtWeights_bkg = "Weight_lumi*%s" ...
        %evtWeights_sig
loader.SetSignalWeightExpression(
          evtWeights_sig)
loader.SetBackgroundWeightExpression(
          evtWeights_bkg)

# Do event selection and train/test split
selStr = "Event_pass_presel_mu && %s" ...
          %Regions[region]
evtSel = ROOT.TCut(selStr.replace("e.", ""))
loader.PrepareTrainingAndTestTree(evtSel,
          "SplitMode=Random:!V")
```

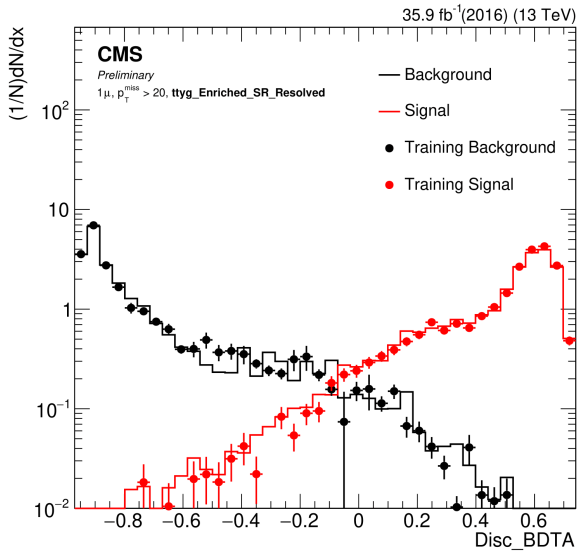
Finally, the classification is done with a TMVA `Factory` object, shown in the code below.

```
m = method # method = "BDTA"
factory = ROOT.TMVA.Factory(
              "TMVA_Classification",
              outputFile,
              "!V:ROC:!Silent:Color: ...
              !DrawProgressBar: ...
              AnalysisType=Classification")

# Book method; params previously defined
factory.BookMethod(loader, methodDict[m][0],
                   m, methodDict[m][1])

# Do classification
factory.TrainAllMethods()
factory.TestAllMethods()
factory.EvaluateAllMethods()
```

In booking the method, "`methodDict`" is an imported dictionary with options for each MVA method. For the BDTA model, the `methodDict` entry is:

```
"BDTA": [ ROOT.TMVA.Types.kBDT,
              "!H:!V:NTrees=850:nCuts=30: ...
              MaxDepth=6: ...
              BoostType=AdaBoost: ...
              AdaBoostBeta=0.05: ...
              UseBaggedBoost: ....
              BaggedSampleFraction=0.5: ...
              SeparationType=GiniIndex" ]
```

The results of the training, testing, and evaluation are an XML file containing the trained model weights and a ROOT file containing several relevant histograms, including the training and testing discriminant distributions and the ROC curve, both shown below.



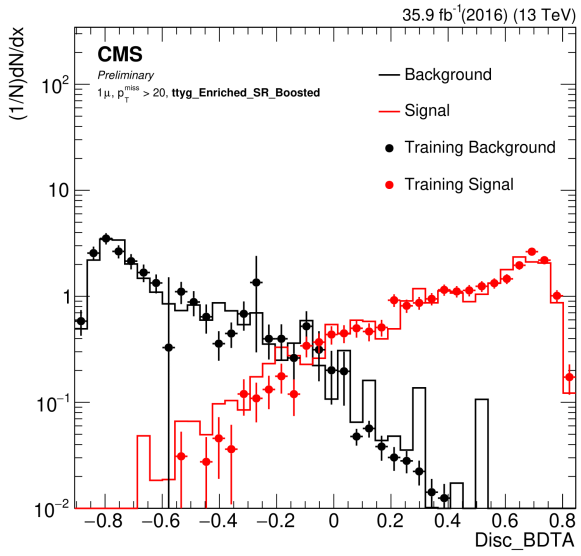
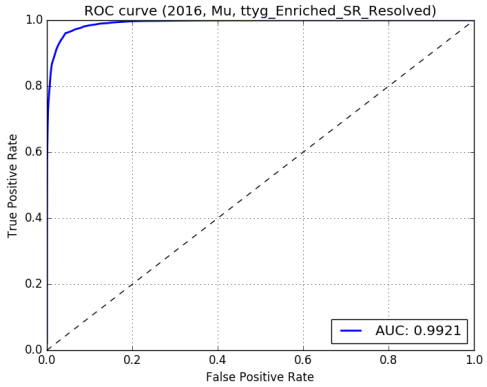


Figure 6: Discriminant histograms for the BDTA model in the resolved (upper plot) and boosted (lower plot) regions. Note that the y-axis is log-scaled.



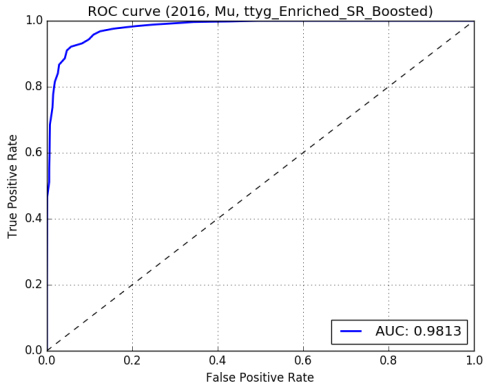


Figure 7: ROC curves for the BDTA model in the resolved (upper) and boosted (lower) regions. The area-under-the-curve (AUC) score for the resolved region was 99.21%; the AUC score for the boosted region was 98.13%.

III XGBoost BDT Model

A BDT model created with XGBoost was briefly considered, but the preliminary results were unpromising, so the model was abandoned. The model and its early results are described here.

The signal and background TTrees were loaded the same way as for the TMVA BDTA model, but for the XGBoost model, the TTrees were converted into NumPy (imported as `np`) arrays using the root-numpy package, as shown below, where "`branchList`" is a list containing the keys of the `varDict` dictionary described previously.

```
sigArr = root_numpy.tree2array(sigTree,
          branches=branchList,
          selection=selStr.replace("e.", ""))

bkgArr = root_numpy.tree2array(bkgTree,
          branches=branchList,
          selection=selStr.replace("e.", ""))
```

The signal and background arrays were combined, reshaping as necessary, into a single array containing all variable information for every event. Labels for the signal and background events were created as an array: `yArr = np.hstack([np.ones(num_sig), np.zeros(num_bkg)])`.

The train/test split was performed by converting the variable and label arrays into Pandas `DataFrame` objects (named `X` and `y`, respectively), then passing these dataframes to Scikit-learn's `train_test_split` function, as shown below.

```
X_train, X_test, y_train, y_test = ...
          train_test_split(X, y,
                           test_size=0.2,
                           random_state=123)
```

Finally, the `XGBClassifier` model was constructed and the fit was performed, as shown below.

```
# Construct model
model = XGBClassifier(max_depth=5,
                      n_estimators=900)

# Train model
model.fit(X_train, y_train)
```

The discriminant distribution was computed by calling the `predict()` method on the trained model, as shown below. The ROC curve and AUC score were computed using Scikit-learn's `roc_curve` and `auc` functions, also shown below.

```
# Test model
y_pred = model.predict(X_test)

# Compute ROC curve
fpr,tpr,thresholds = roc_curve(y_test,y_pred)
AUC = auc(fpr,tpr)
```

Figures 8 and 9 show the discriminant distributions and ROC curves, respectively, for both the resolved and boosted regions. As shown in the ROC curves, the XGBoost BDT model only obtained AUC scores of 87-88%, very low compared to the TMVA BDTA model. The XGBoost BDT was not investigated further due to this poor performance.

Note also that the XGBoost `predict()` method returns a prediction of exactly 0 or 1, no values in between, so the discriminant histograms have only two bins, and the ROC curves have an angular shape.

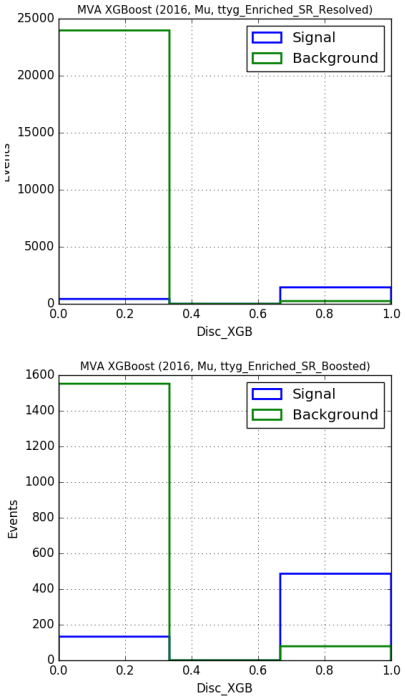


Figure 8: Discriminant histograms for the XGBoost BDT model in the resolved (upper) and boosted (lower) regions.

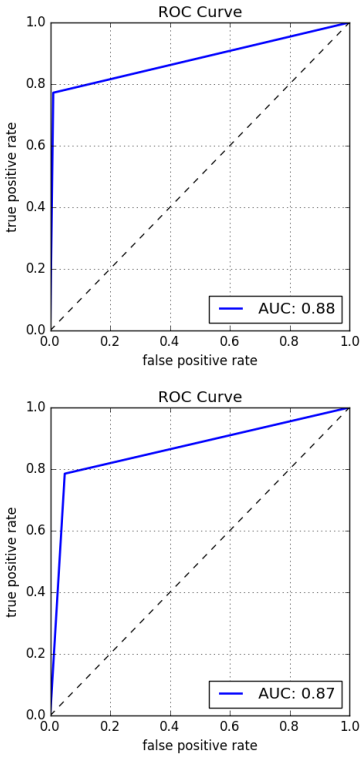


Figure 9: ROC curves for the XGBoost BDT model in the resolved (upper) and boosted (lower) regions.

IV Keras NN Model

Several different NN architectures were explored using the Keras framework within TMVA (implemented as "PyKeras" methods). Using Keras within TMVA allows for later use of the TMVA Reader, which in turn allows for easy calculation of the 95% CL upper limits from the Keras NN model's discriminant. Note that the incompatibility of TMVA and XGBoost is another reason why the XGBoost BDT was ultimately abandoned.

The dataloader steps described in section II were repeated for the Keras NN model. Variable preprocessing was done when booking the method in a TMVA factory; variable decorrelation and Gaussianisation (where the mean of the distribution is normalized to zero, and the width to unity) were performed in that order [2]. When booking the method, similarly to the TMVA BDTA model, parameters are passed from the imported "methodDict" dictionary. For the PyKeras models, the methodDict entry is:

```
"PyKeras": [ROOT.TMVA.Types.kPyKeras,
            "H:!V:VarTransform=D,G: ...
            FilenameModel=kerasModel.h5: ...
            NumEpochs=30:BatchSize=64"]
```

The primary layer type in each Keras model was the "Dense" (densely-connected) layer, where each node of the dense layer connects to every node of the previous layer. Only dense layers were used in the first model. In the second and fifth models, a "Dropout" layer was added, which randomly drops a fraction (in this case, 20%) of the previous layer's nodes in order to prevent overfitting. The optimizer for models 1, 2, 4, and 5 was Stochastic Gradient Descent (SGD); for model 3, the Adam optimizer was investigated. All models used the binary crossentropy loss function, and all were trained over 30 epochs with a batch size of 64. Summaries of each model are shown below.

First (original) model, compiled as:

```
model.compile(loss='binary_crossentropy', optimizer=SGD(lr=0.01), metrics=['accuracy', ])
```

```
Method: PyKeras1
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                1024
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 520
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,562
Trainable params: 1,562
Non-trainable params: 0
```

Second model: Dropout layer added. Compiled the same as first model.

```
Method: PyKeras2
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                1024
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 520
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,562
Trainable params: 1,562
Non-trainable params: 0
```

Third model: original model with Adam optimizer. Compiled as:

```
model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.01), metrics=['accuracy', ])
```

```
Method: PyKeras3
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                1024
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 520
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,562
Trainable params: 1,562
Non-trainable params: 0
```

Fourth model: more nodes per layer. Compiled the same as first model.

```
Method: PyKeras4
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               2048
_________________________________________________________________
dense_2 (Dense)              (None, 32)                4128
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 66
=================================================================
Total params: 6,242
Trainable params: 6,242
Non-trainable params: 0
```

Fifth model: dropout layer with more nodes per layer. Compiled the same as first model.

```
Method: PyKeras5
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 130
=================================================================
Total params: 10,434
Trainable params: 10,434
Non-trainable params: 0
```

Figure 10 shows the ROC curves calculated for each Keras NN model, along with the ROC curve for the BDTA model for comparison, in both the resolved and boosted regions. In the resolved region, all five NN models performed similarly to the BDTA model (AUC: 0.9921). The NN models showed greater variation in the boosted region; the lowest AUC score was 0.7639 for model 5, and the highest AUC score was 0.9521 for model 4. In both regions, the BDTA model outperformed the NN models.

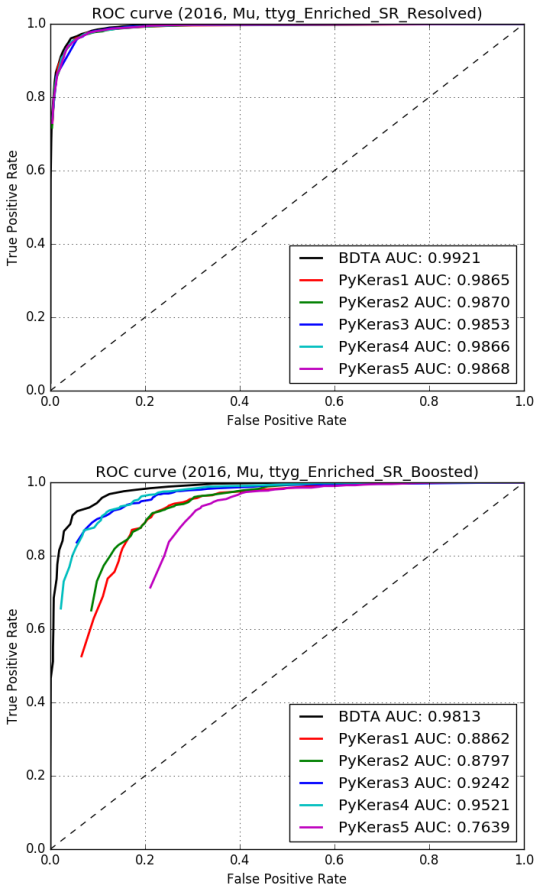


Figure 10: ROC curves for each Keras model, along with the TMVA BDTA model for comparison, in the resolved (upper) and boosted (lower) regions.

Figures 11 through 15 show the training and testing discriminant distributions for each Keras NN model in the resolved and boosted regions. The shapes of the NN discriminant plots are very different from the BDTA discriminant plot; the NN plots show significantly more false negatives and (especially) false positives at the extremes of the histograms compared to the BDTA plot in Figure 6.

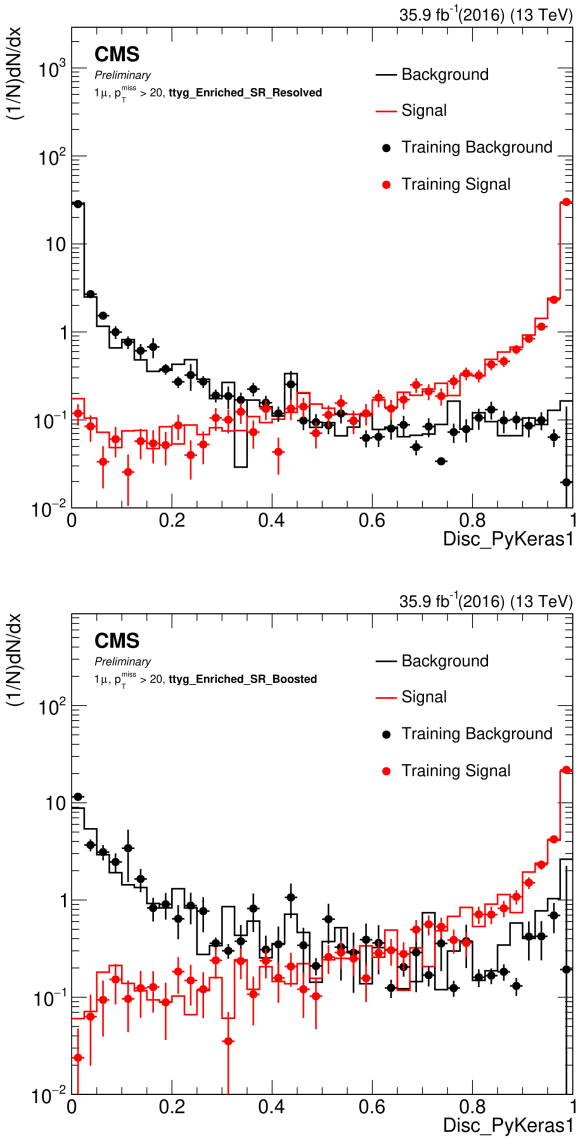


Figure 11: Discriminant histograms for the model `PyKeras1` in the resolved (upper) and boosted (lower) regions. Note that the y-axis is log-scaled.

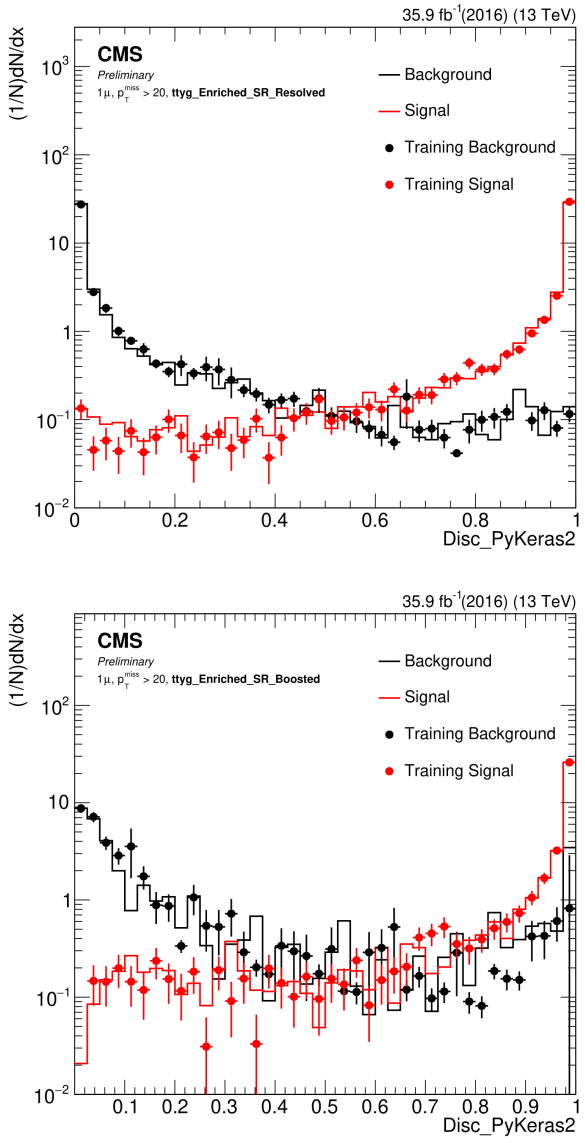


Figure 12: Discriminant histograms for the model `PyKeras2` in the resolved (upper) and boosted (lower) regions. Note that the y-axis is log-scaled.

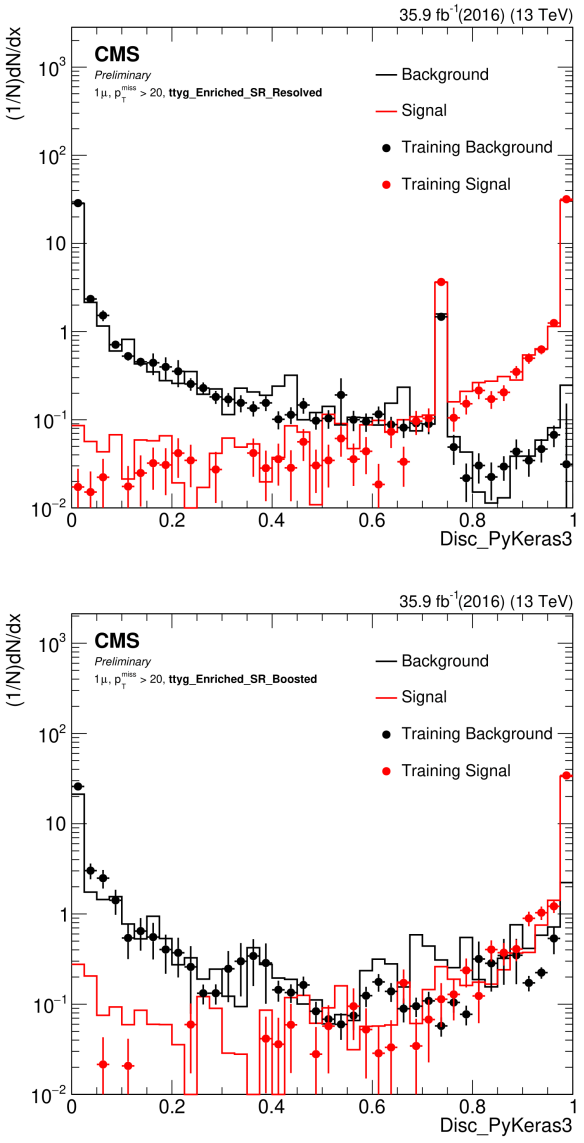
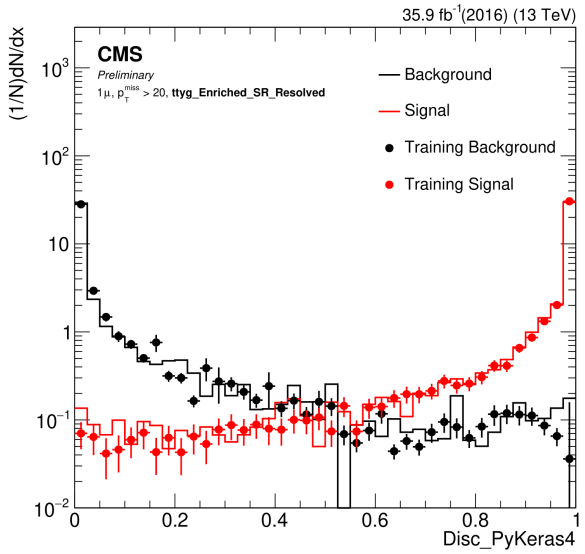
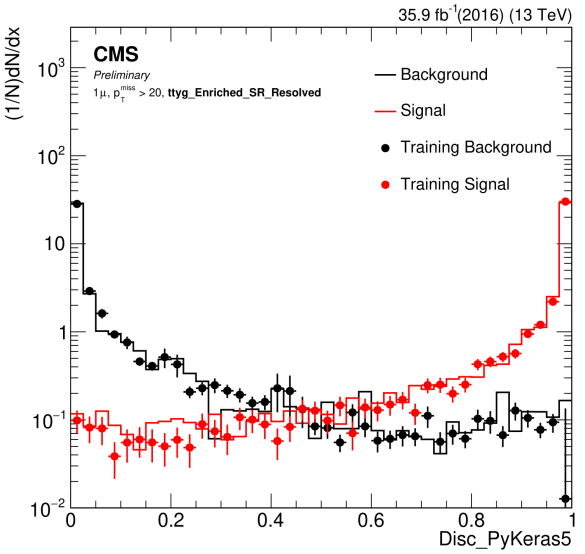
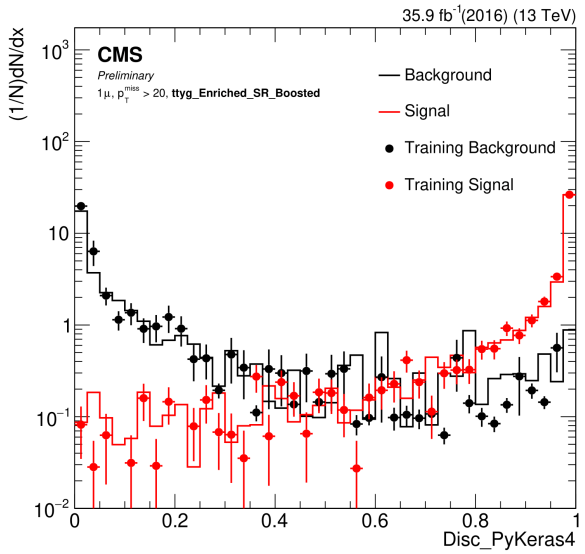


Figure 13: Discriminant histograms for the model `PyKeras3` in the resolved (upper) and boosted (lower) regions. Note that the y-axis is log-scaled.







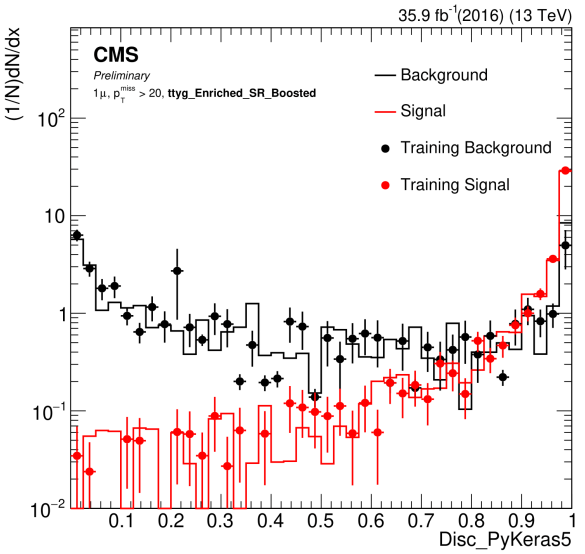


Figure 14: Discriminant histograms for the model `PyKeras4` in the resolved (upper) and boosted (lower) regions. Note that the y-axis is log-scaled.

Figure 15: Discriminant histograms for the model `PyKeras5` in the resolved (upper) and boosted (lower) regions. Note that the y-axis is log-scaled.

V Comparison of Results

In the search for t^* events, an upper limit on the signal strength modifier (represented as a product of the signal cross section $\sigma_{t^*\bar{t}^*}$ and the branching ratios $\mathcal{B}(t^* \to tg)\mathcal{B}(t^* \to t\gamma)$ of the target decay mode) at 95% confidence level is computed for several mass points m_{t^*} [3]. The region where the observed product $\sigma \times \mathcal{BB}$ is less than the product predicted by theory is excluded, i.e., the mass m_{t^*} is expected to be greater than the point where the theory and MVA plots in Figure 16 intersect. Since the theory curve in Figure 16 decreases with m_{t^*}, the better upper limit corresponds to the curve with smallest $\sigma \times \mathcal{BB}$ at each mass point.

The limits computed from the six MVA discriminants (one TMVA BDTA model and five Keras NN models) are compared to the limits computed from the `Reco_mass_T` and `Reco_st` variables alone. The tables in Figure 17 show the improvement of each MVA discriminant over `Reco_mass_T` in the resolved and boosted regions. From the limit plots in Figure 16, it is obvious that the TMVA BDTA model improved over both the `Reco_mass_T` and `Reco_st` variables as well as the Keras NN discriminants in both regions for all mass points.

As for the Keras NN models, in the resolved region, every model except for `PyKeras3` improved over `Reco_mass_T` up to mass $m_{t^*} = 1500\ GeV$, but no model improved over `Reco_st`. In this region, the model `PyKeras2` produced the best limits, but models 1, 2, 4, and 5 all gave similar results.

In the boosted region, all models performed considerably worse than in the resolved region, as expected from the poor AUC scores in the boosted-region ROC curves (Figure 10). Model `PyKeras5`, for example, nowhere improved over `Reco_mass_T`. The best limits were produced by model `PyKeras4`, but even this model did not improve over `Reco_mass_T` above mass $m_{t^*} = 1300\ GeV$. Although other NN model architectures could be investigated, it is not expected that these models will outperform the BDTA.

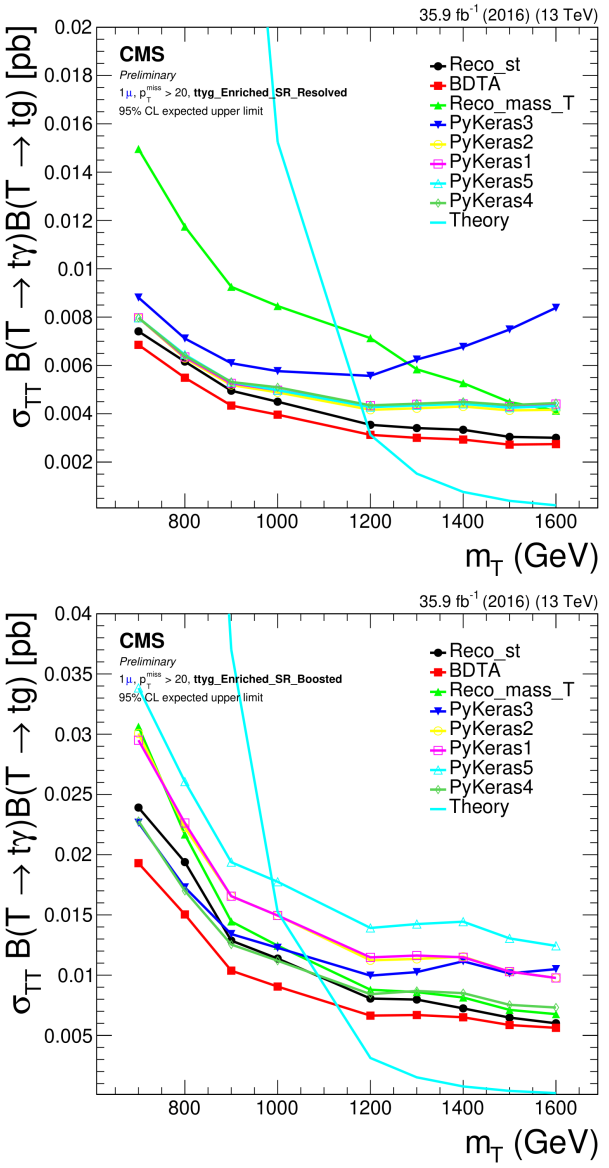


Figure 16: 95% CL upper limit plots for `Reco_mass_T`, `Reco_st`, BDTA, and all NN models in the resolved (upper) and boosted (lower) regions over a range of mass points m_{t^*}. The limit predicted from theory is also shown for comparison with the observed limits.

2016, μ, ttyg_Enriched_SR_Resolved

mT	Reco_mass_T	BDTA	PyKeras1	PyKeras2	PyKeras3	PyKeras4	PyKeras5	Reco_st
700.0	0.014960	-54.2	-46.7	-46.7	-41.1	-46.7	-46.7	-50.5
800.0	0.011745	-53.3	-45.9	-46.3	-39.4	-45.5	-45.1	-47.6
900.0	0.009254	-53.1	-43.2	-43.9	-34.2	-42.6	-42.6	-46.5
1000.0	0.008458	-53.2	-41.2	-42.3	-31.9	-39.8	-41.2	-46.8
1200.0	0.007130	-56.2	-39.2	-41.6	-21.9	-39.0	-40.1	-50.3
1300.0	0.005839	-48.6	-24.8	-27.6	6.9	-24.4	-25.6	-41.7
1400.0	0.005265	-44.3	-16.3	-18.3	28.5	-14.7	-16.3	-36.7
1500.0	0.004482	-39.3	-4.1	-7.7	67.0	-2.7	-5.2	-32.1
1600.0	0.004127	-33.5	6.6	0.9	103.2	7.6	4.7	-27.2

2016, μ, ttyg_Enriched_SR_Boosted

mT	Reco_mass_T	BDTA	PyKeras1	PyKeras2	PyKeras3	PyKeras4	PyKeras5	Reco_st
700.0	0.030620	-37.0	-3.7	-1.8	-26.0	-25.6	10.5	-21.9
800.0	0.021675	-30.6	4.4	2.6	-20.3	-21.6	20.3	-10.6
900.0	0.014459	-28.3	14.5	14.5	-7.3	-13.2	34.0	-11.0
1000.0	0.012449	-27.3	20.1	20.1	-1.4	-10.0	42.6	-8.6
1200.0	0.008790	-24.4	30.6	27.8	13.3	-4.2	58.3	-8.3
1300.0	0.008592	-22.1	35.4	32.0	19.3	1.1	65.7	-7.2
1400.0	0.008148	-20.2	40.9	41.5	36.8	4.4	77.2	-11.1
1500.0	0.007117	-17.6	44.6	44.6	42.6	5.9	83.4	-9.0
1600.0	0.006764	-16.8	44.4	44.4	55.2	8.1	83.8	-11.2

Figure 17: 95% CL upper limit improvements over `Reco_mass_T` for `Reco_st`, BDTA, and all NN models in the resolved (upper) and boosted (lower) regions. A value of, e.g., -37.0 for BDTA at mT = 700 (GeV implied) means the BDTA model's limit was 37% improved over the limit computed from `Reco_mass_T`, which is shown in the second column.

References

[1] CMS Collaboration. Search for pair production of excited top quarks in the lepton+jets final state. *Phys. Lett. B*, 778:349–370, 2018. ISSN 0370–2693. doi: 10.1016/j.physletb.2018.01.049.

[2] A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. von Toerne, H. Voss, M. Backes, T. Carli, O. Cohen, A. Christov, D. Dannheim, K. Danielowski, S. Henrot-Versille, M. Jachowski, K. Kraszewski, A. Krasznahorkay, M. Kruk, Y. Mahalalel, R. Ospanov, X. Prudent, A. Robert, D. Schouten, F. Tegenfeldt, A. Voigt, K. Voss, M. Wolter, and A. Zemla. TMVA - Toolkit for Multivariate Data Analysis. 2007. doi: 10.48550/ARXIV.PHYSICS/0703039.

[3] R. K. Verma. cms-TT-run2, 2022. URL `https://github.com/ravindkv/cms-TT-run2`.