Quark-Gluon Jet Discrimination: Can a classifier find the exact mixtures?
Florida State University

Ali Al Kadhim

December 12, 2019

Abstract

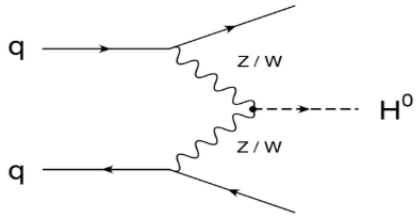
Contents

Chapter 1

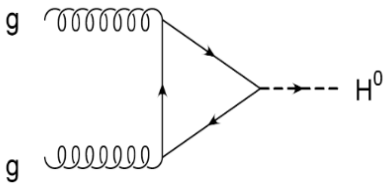
Introduction and Physics Motivation

1.1 Introduction

The Large Hadron Collider produces a billion proton-proton collision a second, each of which having one or several jets. Although this is an ambitious goal, Understanding these jets in great detail could be one of the most paramount endeavours in unravelling physics beyond the standard model. The identification the origin of jets, i.e. whether they are quark or gluon jets, is an extremely important experimental test in uncovering the fundamental physics that occurs in a given event. Of particular interest to this report, is the implications that quark-gluon jet identification could have on understanding the properties of the Higgs boson, since the Higgs properties and couplings rely on the weak-boson-fusion production process $qq \to Hqq$. For example, if one wants to measure the Higgs boson coupling to gauge bosons, one wouldj look at this process and not the gluon-fusion process $gg \to Hgg$, which is the more frequent process.



(a) Higgs from Vector Boson Fusion



(b) Higgs from Gluon Fusion

Chapter 2

Data Exploration

2.1 Description of the Dataset

2.1.1 Important Definitions

Before we begin describing the datasets used, we must review some important definitions and facts regarding the experimental aparatus. The CMS experiment uses a right-handed coordinate system, with the origin at the nominal interaction point, the x axis pointing to the center of the LHC ring, the y axis pointing up, and the z axis along the counterclockwise-beam direction as viewed from above (see figure below). θ is measured from the positive z axis and the azimuthat angle ϕ is measure in the $x-y$ plane. The pseudorapidity $\eta = -\ln[\tan(\theta/2)]$ and the rapidity $y = \ln\left[(E+p_z)\,/\,(E-p_z)\right]$ are equal for massless particles. f

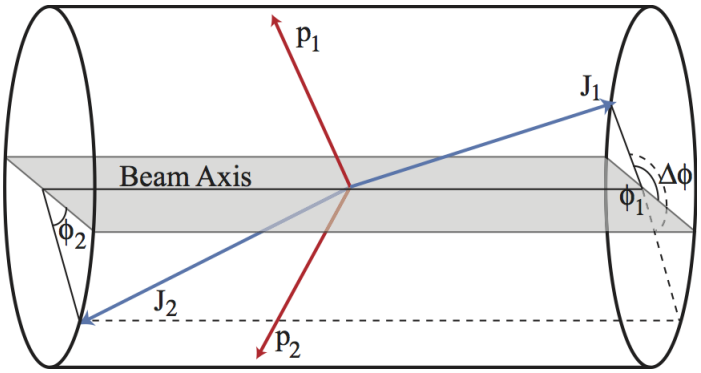


Figure 2.1: The used Dataset from CMS Open Data

The dataset used for this report [1] is provided by CMS Open Data uses Monte Carlo simulated proton-proton collisions at 13 TeV in order to extract particle jets, which are reconstructed from the particle-flow (PF) algorithm. The reconstructed

particles (also called PF candidates) were reconstructed from the simulated detector signal. For each jet, there are variables describing the jet on a high-level, particle level, as well as generator-level. There are also variables describing the collision event and conditions of its simulation. The dataset is derived from MINIAODSIM simulated data, using Pythia 8, which correspond to from collision data collected by the CMS experiment in 2016. These AOD filed were then converted to hdf5 n-tuples, and then to csv/dataframe n-tuples for easy analysis and visualization. This 3-level hierarchical data means that each row in the dataset contains all of the info pertaining to that jet.

Figure 2.2: The used Dataset from CMS Open Data

The jets in this dataset were clustered from the PF candidates of each collision event using the antti-k_t algorithm with $R = 0.4$. For each collision event, only those jets with transverse momentum exceeding 30 GeV, and pseudorapadity less than 2.5 were saved to file. The resulting jet flavor is obtained from the generator level particles by a jet flavor algorithm, which attempts to match a reconstructed jet to a single initiating particle. The reconstructed (light, i.e. u, d, s) jet flavor, which we are interested in studying, can be found in the 'parton' and 'physics' definition variables. Since the 'parton' definition of light jet flavors is biased towards the b- and c- quarks, we use they physics definition of jet flavors instead.
Important definitions Partons (except for heavy quarks) are essentially massless. Jets, in particular those with significant substructure, are not. Jet masses are interesting in part because hadronic decays of very high-pt top quarks and electroweak bosons will be collimated by the Lorentz boost factor and so form a single jet, whose invariant mass might provide a means to identify the origin of the jet.

The immediate goal of this project is to explore the potential of advanced classi cation methods to improve the statistical signi

cance of the experiment.

The dataset contains 100000 jets. The dataset was reduced to include only quark and gluon flavor jets. consists of 66 Features (the full features are included in Ref. [**bib:Data**] for reference). However, not all of these features are useful or even pertinent for our task.

Since we are only interested in the reconstructed jet flavor we only explore the high-level variables. We also can explore the features further by studying their feature importance. We want to only include features that are useful for our classification task (i.e. the most important features) to train our model. This ensures that we reduce the (training) times and computational cimplexity, as well as reduce the model complexity (we should choose the least amount of features that produce the best predictions in order to avoid overfitting). After the genJet and PF features were removed, exploring the importance of the remaining features was done by random forest (detailed explanation in section). This can be seen in figure

2.2 Data Exploration

And also can be seen in boxplots, where we see that features jetQGl have the least overlap, i.e. the most important features for our classification task

The distributions of these features over the whole range are plotted in figure

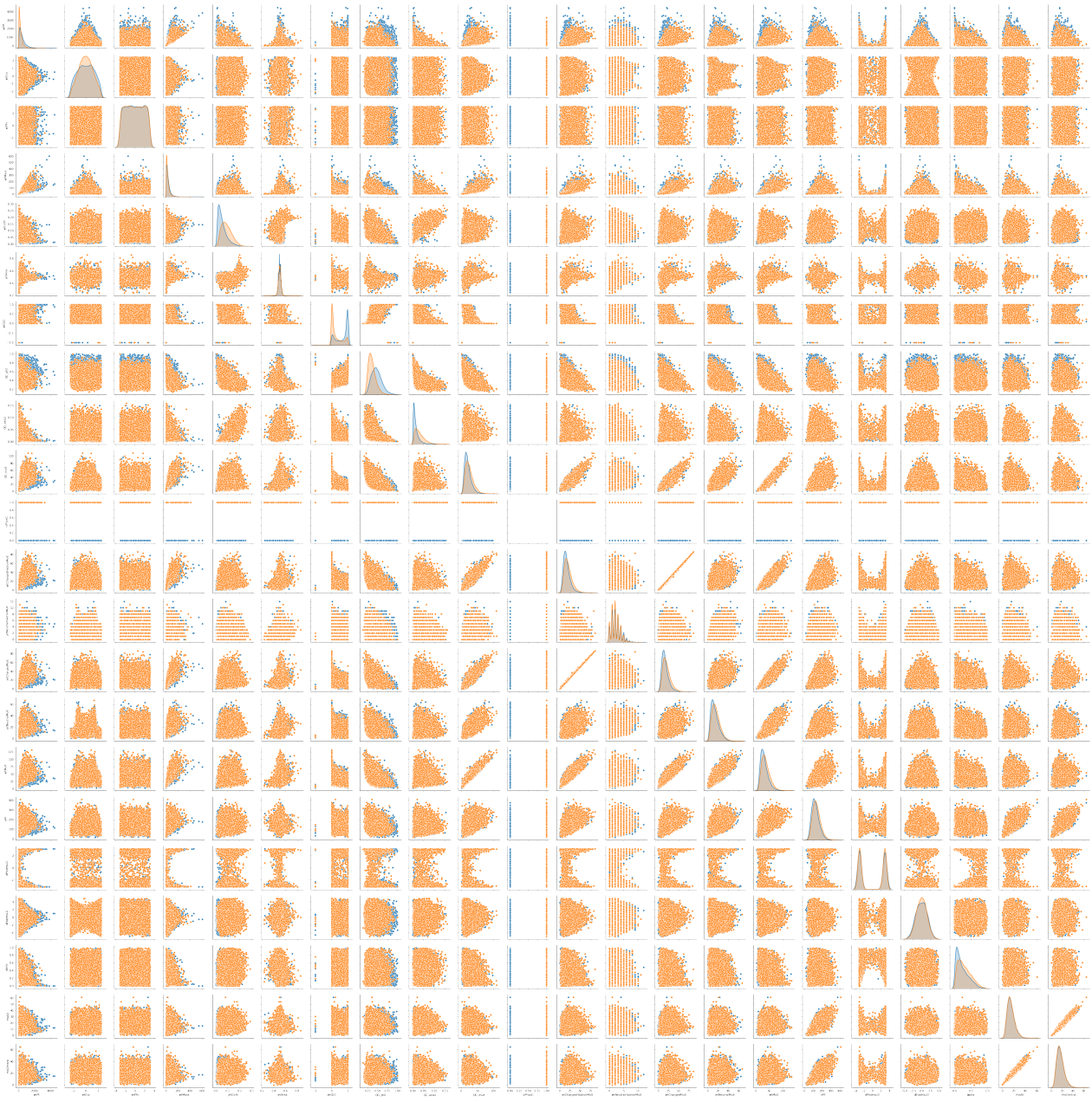


Figure 2.3: complete Features pairplot

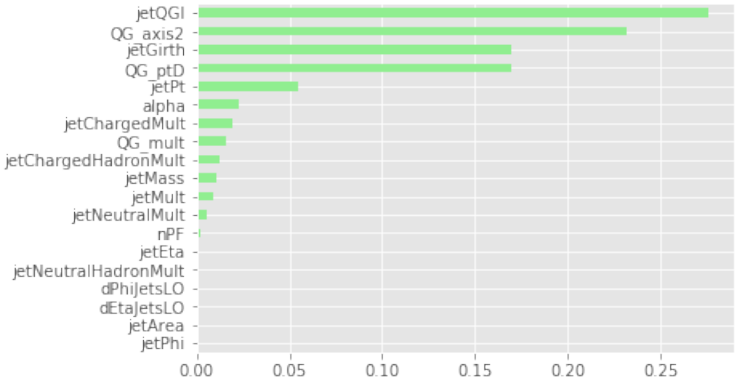


Figure 2.4: Feature Importance

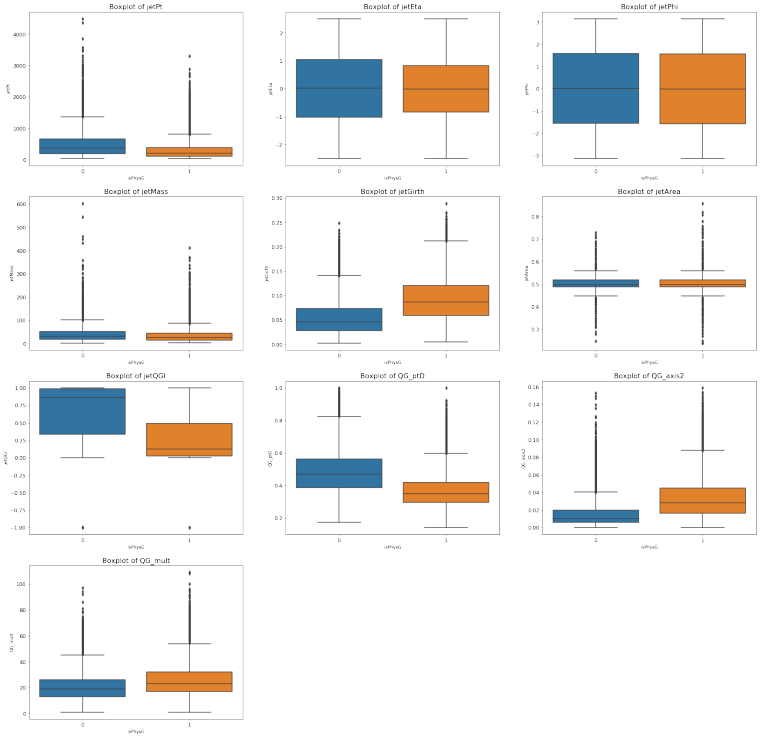
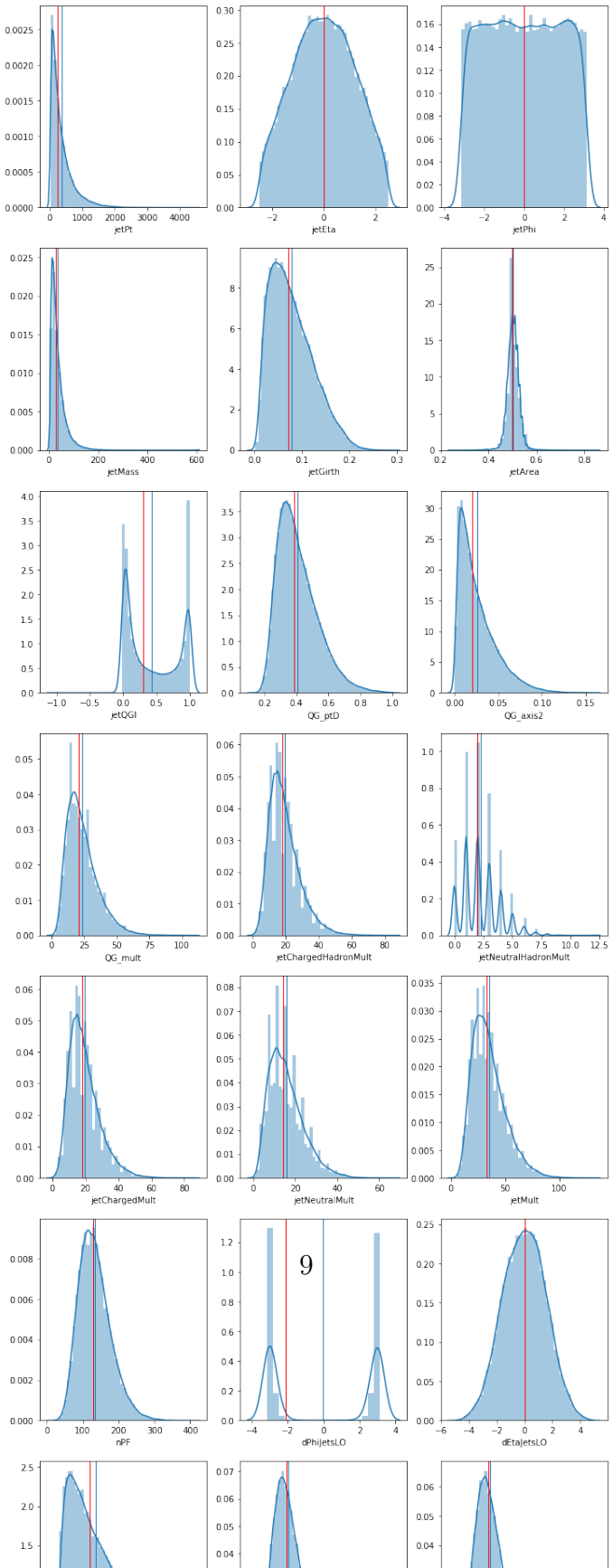


Figure 2.5: Important Features Box Plots

Distribution of Features



Chapter 3

Learning to Classify: Testing Different Classifiers

3.1 Machine Learning Basics

Before we start, we standarize the data, i.e. transform $x^{(i)}$ to $x^{(i)} = \frac{x^{(i)}-\mu(x)}{\sigma(x)}$.

To establish notation, I use $x^{(i)}$ to denote features (or input variables, i.e. the data that we are training the classifier on), and $y^{(i)}$ to mean the target variable (in our case, the target is either 1 or 0, where 1 = gluon jet, and 0 = quark jet). A pair $(x^{(i)}, y^{(i)})$ is called a trainng example, and the dataset that we use has N examples $\{(x^{(i)}, y^{(i)}); i = 1, 2, ..., N$ is called the trainig set. In general, capitalized (X, Y) mean the space of input variables (data) or output variables (prediction). To perform supervised learning, we decide to approximated y as a linear function f of x:

$$f_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \tag{3.1}$$

Here, the θ_i's are the parameters (sometimes called weights) parameterizing the space of functions mapping $X \to Y$. We can drop this subscript since the only variable f depends on is x and θ are just multiplicative constants. Then

$$f(x) = \sum_{i=0}^{N} \theta_i x_i = \theta^T x \tag{3.2}$$

Where on the right hand side we represented them are vectors. In order to learn the parameter θ we need to see how close our function is (which is trying to approximated y) to the y. To do this, we define a function that measures, for each value of θ, how close the $f(x^{(i)})$'s are to the corresponding $y^{(i)}$'s. This is defined a

the cost function:

$$J(\theta) = \sum_{i=1}^{m} \left(f_\theta \left(x^{(i)} \right) - y^{(i)} \right)^2 \tag{3.3}$$

Which can be just viewed as the Euclidean distance from f to y. Now we want to choose θ that minimizes $J(\theta)$. To do that, we start with an initil guess for θ and repeatedly make $J(\theta)$ smalled for each θ until it converges to a value of θ that minimizes $J(\theta)$. We consider the popular gradient descent algorithm, which starts with a θ and repeatedly updates it:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{3.4}$$

where α is called the learning rates. This algorithm repeatedly takes a step towards the direction of the steepest decrease of J. The partial derivative on the right hand side can be calculated

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(f_\theta(x) - y \right)^2 \\ &= 2 \cdot \frac{1}{2} \left(f_\theta(x) - y \right) \cdot \frac{\partial}{\partial \theta_j} \left(f_\theta(x) - y \right) \\ &= \left(f_\theta(x) - y \right) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\ &= \left(f_\theta(x) - y \right) x_j \end{aligned} \tag{3.5}$$

Hence for a single training example we have the least0mean squares update rule

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - f_\theta \left(x^{(i)} \right) \right) x_j^{(i)} \tag{3.6}$$

Therefore if we if we are encountering a training example on which our prediction matches closes to y then little is needed to change the parameters, so it converges. On the otherhand, a larger change in parameters is needed if our prediction f is very far off from y. Hence we repeat this update rule above, updating the parameters until we read convergence.
Let's assme that the target variables and the input variables are related by the equation

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)} \tag{3.7}$$

Where $\epsilon^{(i)}$ is an error term or random noise. Furthermore, we know very little about these error terms, hence it should be modeled as a random variable itself, assuming they are iid (independent and identically distributed) according to a normal distribution with mean zero and some variance σ^2, so the density of $\epsilon^{(i)}$ is

$$p\left(\epsilon^{(i)}\right) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(\epsilon^{(i)}\right)^2}{2\sigma^2} \right) \tag{3.8}$$

Wich implies that

$$p\left(y^{(i)}|x^{(i)};\theta\right) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\left(y^{(i)}-\theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \tag{3.9}$$

This quantity is typically viewed as a function of Y for a fixed value of θ. When we wish to explicity view this as a function of θ, we call it the likelihood function:

$$L(\theta) = L(\theta; X, Y) = p(Y|X;\theta) \tag{3.10}$$

Which can be written as

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{N} p\left(y^{(i)}|x^{(i)};\theta\right) \\ &= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\left(y^{(i)}-\theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \end{aligned} \tag{3.11}$$

We wish to approximate θ by the prinicple of maximum likelihood, which says that we should choose θ so as to make the data as high probability as possible, i.e. we should θ to maximize $L(\theta)$. However, it is simpler if we maximize the log likelihood $l(\theta)$:

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \log \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\left(y^{(i)}-\theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^{N} \log \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\left(y^{(i)}-\theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \\ &= N\log\frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2}\cdot\frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)}-\theta^T x^{(i)}\right)^2 \end{aligned} \tag{3.12}$$

So maximizing $l(\theta)$ gives the same answer as minimizing

$$\frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)}-\theta^T x^{(i)}\right)^2 \tag{3.13}$$

which is $J(\theta)$, our original cost function.

For our case, we have the problem of binary classification, so using Baye's Theorem, the probability of y given x is given by $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$, where in our

case, we have binary classification where $y = 1$ is the positive class, i.e. the jet is a gluon jet, or $y = 0$ which is the negative class, i.e. the jet is a quark jet. So the probability that we have a gluon jet (signal) is

$$\begin{aligned} P(y=1|x) &= P(gluon|x) \\ &= \frac{p(x|y=1)p(y=1)}{p(x|y=1)p(y=1)+p(x|y=0)p(y=0)} \\ &= \frac{p(x|gluon)p(gluon)}{p(x|gluon)p(gluon)+p(x|quark)p(quark)} \end{aligned} \tag{3.14}$$

3.2 Starting Simple: Logistic Regression, Quadratic Discriminant Analysis, and Naive Bayes

3.2.1 Logistic Regression

We know that $y \in 0, 1$, so we choose

$$f_\theta(x) = g\left(\theta^T x\right) = \frac{1}{1+e^{-\theta^T x}} \tag{3.15}$$

where $g(z) = \frac{1}{1+e^{-z}}$ is called the sigmoid function, which is a smoothly increasing function from 0 to 1. For this binary classification, we can assume

$$\begin{aligned} P(y=1|x;\theta) &= f_\theta(x) \\ P(y=0|x;\theta) &= 1 - f_\theta(x) \end{aligned} \tag{3.16}$$

which can be written as

$$p(y|x;\theta) = \left(f_\theta(x)\right)^y \left(1-f_\theta(x)\right)^{1-y} \tag{3.17}$$

So for N training examples, we have

$$\begin{aligned} L(\theta) &= p(\vec{y}|X;\theta) \\ &= \prod_{i=1}^{N} p\left(y^{(i)}|x^{(i)};\theta\right) \\ &= \prod_{i=1}^{N} \left(f_\theta\left(x^{(i)}\right)\right)^{y^{(i)}} \left(1-f_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}} \end{aligned} \tag{3.18}$$

And our log likelihood

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{N} y^{(i)} \log f\left(x^{(i)}\right) + \left(1-y^{(i)}\right) \log\left(1-f\left(x^{(i)}\right)\right) \end{aligned} \tag{3.19}$$

And we can maximize the log likelihood by gradient descent described earlier, where now we can use a vector notation so the step is given by $\theta = \theta + \alpha \nabla_\theta \ell(\theta)$. Note that the derivative of the sigmoid function is

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{\left(1+e^{-z}\right)^2}\left(e^{-z}\right) \\ &= \frac{1}{\left(1+e^{-z}\right)^2}\cdot\left(1-\frac{1}{\left(1+e^{-z}\right)}\right) \\ &= g(z)(1-g(z)) \end{aligned} \tag{3.20}$$

so that the derivative for each step is:

$$\begin{aligned} \frac{\partial}{\partial\theta_j}\ell(\theta) &= \left(y\frac{1}{g\left(\theta^T x\right)} - (1-y)\frac{1}{1-g\left(\theta^T x\right)}\right)\frac{\partial}{\partial\theta_j} g\left(\theta^T x\right) \\ &= \left(y\frac{1}{g\left(\theta^T x\right)} - (1-y)\frac{1}{1-g\left(\theta^T x\right)}\right) g\left(\theta^T x\right)\left(1-g\left(\theta^T x\right)\right)\frac{\partial}{\partial\theta_j}\theta^T x \\ &= \left(y\left(1-g\left(\theta^T x\right)\right) - (1-y)g\left(\theta^T x\right)\right)x_j \\ &= \left(y - f_\theta(x)\right)x_j \end{aligned} \tag{3.21}$$

This gives the stochastic gradient ascent given logistic regression, where in each step, the following is calculated

$$\theta_j = \theta_j + \alpha\left(y^{(i)} - f_\theta\left(x^{(i)}\right)\right)x_j^{(i)} \tag{3.22}$$

Since we have $y \in \{0,1\}$ y can either be 0 or 1. So it is natural to model y as a Bernoulli $y|x;\theta \sim$ Bernoulli (ϕ) for some appropriate definitions of μ and ϕ as functions of x and ϕ. So $p(y=1;\phi) = \phi; p(y=0;\phi) = 1-\phi$ and the distribution of y can be written as

$$\begin{aligned} p(y;\phi) &= \phi^y(1-\phi)^{1-y} \\ &= \exp(y\log\phi + (1-y)\log(1-\phi)) \\ &= \exp\left(\left(\log\left(\frac{\phi}{1-\phi}\right)\right)y + \log(1-\phi)\right) \end{aligned} \tag{3.23}$$

So for our logistic Regression ordinary least squares, we get

$$\begin{aligned} h_\theta(x) &= E[y|x;\theta] \\ &= \phi \\ &= 1/\left(1+e^{-\eta}\right) \\ &= 1/\left(1+e^{-\theta^T x}\right) \end{aligned} \tag{3.24}$$

And our log likelihood becomes

$$\begin{aligned}\ell(\theta) &= \sum_{i=1}^{m} \log p\left(y^{(i)}|x^{(i)};\theta\right) \\ &= \sum_{i=1}^{m} \log \prod_{l=1}^{k} \left(\frac{e^{\theta_i^T x^{(i)}}}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \right)^{1\left\{y^{(i)}=l\right\}}\end{aligned} \tag{3.25}$$

And as explained above, we then maximize this l in terms of ϕ using gradient ascent. This lengthy derivation is given in a few short lines of code using sklearn, such as in the following figure The untuned model was fit to the data, and the

```
import pandas as pd
import sklearn
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pylab as pl
df = pd.read_csv(r'D:\new datasets1\write up\JetNtuple_RunIISummer16_13TeV_MC_csv_100k_2.csv')
dfqg = df[(df.isPhysG==1) | (df.isPhysUDS==1)].reset_index()
dfqg=dfqg.drop(['index','EventNum' ], axis=1)
from sklearn.model_selection import train_test_split
SEED=12
train, test = train_test_split(dfqg, test_size=0.2, random_state=SEED)
train_y = train.isPhysG
test_y = test.isPhysG
from sklearn.metrics import roc_curve, classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(train_x, train_y)
y_predlog = logreg.predict(test_x)
print(classification_report(test_y, y_predlog))
fpr, tpr, thresholds = roc_curve(test_y, y_predlog, pos_label=1)
plt.plot([0,1], [0,1], 'k--')
plt.plot(fpr, tpr, label = 'Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve')
plt.show()
from sklearn.model_selection import GridSearchCV
penalty = ['l1', 'l2']
C = np.logspace(0, 4, 10) # regularization hyperparamter space
# Create hyperparameter options
hyperparameters = dict(C=C, penalty=penalty)
clf = GridSearchCV(logreg, hyperparameters, cv=5, verbose=0)
best_model = clf.fit(train_x, train_y)
```

Figure 3.1: Sample Code for Logistic Regression Classifier

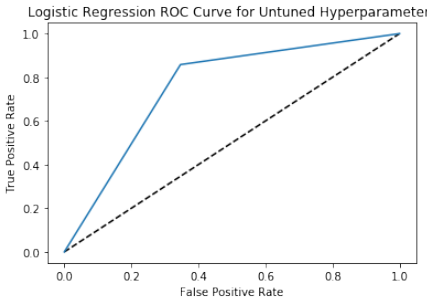
corresponding ROC curve is plotted. Then, the model hyperparameters were tuned with grid search cross validation and (Best Penalty: l2, Best C: 21.5) and the ROC curve with tuned hyperparamers is plotted along with the classification report and confusion matrix

Tuned Logistic Regression Parameters: 'C': 31.622776601683793 We see that the logitic regression classifier does not give good accuracy results, even when tuned.

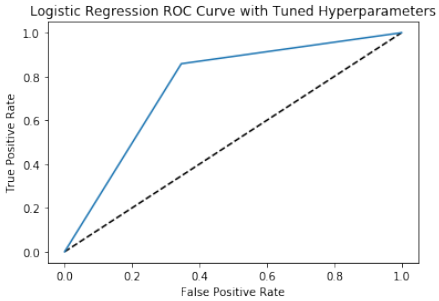
3.2.2 Quadratic Discriminant Analysis

As we saw, logistc regression regression assumes that the observations within each class are drawn from a multivariate Gaussian distribution with a class-specific mean vector $\mu \in R^n$. Quadratic discriminant analysis provides an alternative

centering



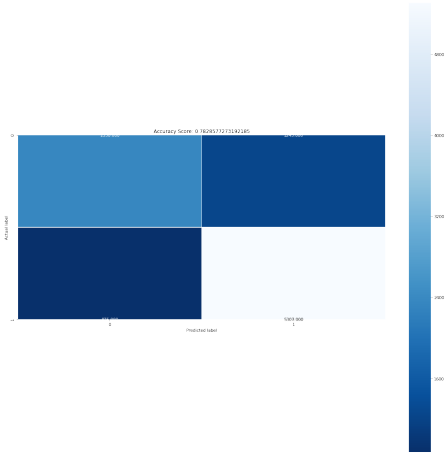
(a) Untuned Logitic Regression Classifier



(b) Tuned Logitic Regression Classifier

	precision	recall	f1-score	support
0	0.86	0.97	0.91	32726
1	0.70	0.32	0.44	7274
accuracy			0.85	40000
macro avg	0.78	0.64	0.68	40000
weighted avg	0.84	0.85	0.83	40000

(c) Accuracy Report for Logistic Regression



(d) Confusion Matric for Logistic Regression Classifier

Figure 3.2: Logistic Regression

approach by assuming that each class has its own covariance matrix $\Sigma \in R^{n \times n}$. So the density is

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right) \tag{3.26}$$

the mean is given by μ

$$\mathrm{E}[X] = \int_x x p(x; \mu, \Sigma) dx = \mu$$

and the covarance of a random variable is defined as $\mathrm{E}\left[(Z - \mathrm{E}[Z])(Z - \mathrm{E}[Z])^T\right]$. The distributions become

$$\begin{aligned} p(y) &= \phi^y (1-\phi)^{1-y} \\ p(x|y=0) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}\left(x-\mu_0\right)^T \Sigma^{-1}\left(x-\mu_0\right)\right) \\ p(x|y=1) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}\left(x-\mu_1\right)^T \Sigma^{-1}\left(x-\mu_1\right)\right) \end{aligned} \tag{3.27}$$

And the log likelihood of the data becomes

$$\begin{aligned} \ell\left(\phi, \mu_0, \mu_1, \Sigma\right) &= \log \prod_{i=1}^{m} p\left(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma\right) \\ &= \log \prod_{i=1}^{m} p\left(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma\right) p\left(y^{(i)}; \phi\right) \end{aligned} \tag{3.28}$$

3.2.3 Naive Baye's Classifier

Since the values for x are discrete we can assume that the x_i's are conditionally dependent on y. This is called the Naive Bayes assumption, and the resulting algorithm is the Naive Bayes Classifier. This means that the knowledge of the values of the data x_i will have no effect on the beliefs about the value of some other data x_j, i.e

$$\begin{aligned} &p\left(x_1, \ldots, x_N|y\right) \\ &\quad = p\left(x_1|y\right) p\left(x_2|y, x_1\right) p\left(x_3|y, x_1, x_2\right) \cdots p\left(x_N|y, x_1, \ldots, x_{N-1}\right) \\ &= p\left(x_1|y\right) p\left(x_2|y\right) p\left(x_3|y\right) \cdots p\left(x_N|y\right) \\ &= \textstyle\prod_{j=1}^{n} p\left(x_j|y\right) \end{aligned} \tag{3.29}$$

Hence we can write the joint likelihood is simple the joint likelihood of the data

$$\mathcal{L} = \prod_{i=1}^{m} p\left(x^{(i)}, y^{(i)}\right) \tag{3.30}$$

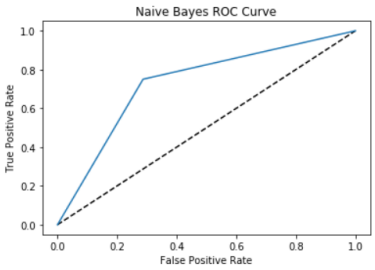
So to make a prediction on a new example with featuress x we calculate

$$p(y=1|x) = \frac{p(x|y=1)p(y=1)}{p(x)} = \frac{\left(\prod_{j=1}^{n} p\left(x_j|y=1\right)\right) p(y=1)}{\left(\prod_{j=1}^{n} p\left(x_j|y=1\right)\right) p(y=1) + \left(\prod_{j=1}^{n} p\left(x_j|y=0\right)\right) p(y=0)} \tag{3.31}$$

The Classification report and ROC curve for Naive Baysian Classifier is plotted

	precision	recall	f1-score	support
0	0.62	0.71	0.67	3596
1	0.82	0.75	0.78	6181
accuracy			0.74	9777
macro avg	0.72	0.73	0.72	9777
weighted avg	0.75	0.74	0.74	9777

(a) Naive Bayes Classifier Classification Report



(b) Naive Bayes Classifier ROC curve

Figure 3.3: Naive Bayes

3.3 K Nearest Neighnors

3.4 Tree Classifiers

3.4.1 Boosting

The idea of boosting was to combine the outputs of many "weak" classifiers to produce a powerful "committee". Consider a

3.5 Classifier Comparison with Tuned Hyperparameters

We measure model performance through accuracy, precision, recall, and f2, where accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$, Precision $= \frac{TP}{TP+FP}$, Recall $= \frac{TP}{TP+FN}$, where

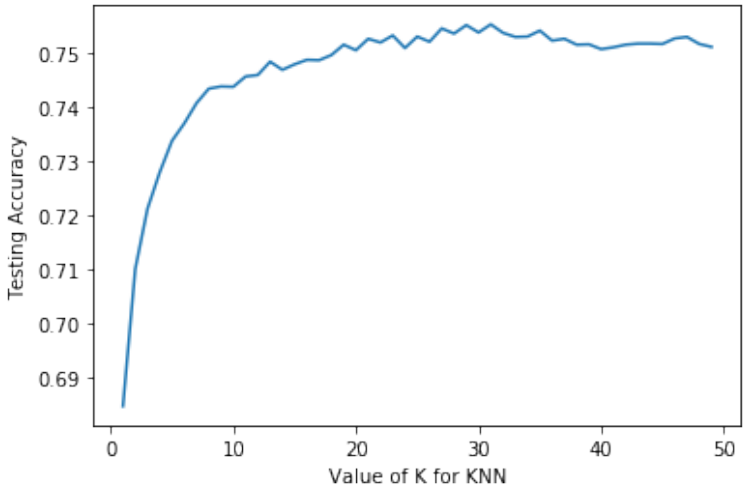


Figure 3.4: Convergence of the Number of Neighbors in KNN classifier

TP: True Positive: Predicted values correctly predicted as actual positive
FP: Predicted values incorrectly predicted an actual positive. i.e., Negative values predicted as positive
FN: False Negative: Positive values predicted as negative
TN: True Negative: Predicted values correctly predicted as an actual negative

The ROC curves for the Untuned tree/graph based classifiers are plotted

Next we perform hyperparameter tuning for each of the classifiers using Randomized Grid Search, which is not computationally expensive to tune some of the model parameters. We then perform Cross Validation Grid Search in order to tune the classifier model parameters more precisely. For Decision Tree Classifier, the best model parapeters were found to be:

$$criterion =' entropy', max_depth = 3, max_features = 6, min_samples_leaf = 1$$

, with the resuluting best score of 0.846.

Best Hyperparameters for Random Forest Classifier:

$$'max_depth' : 4,' max_features' :' log2',' min_samples_leaf' : 0.1,' n_estimators' : 400$$

Implementing the values attained for the tuned hyperparameters (mentioned above), we can immediately plot the ROC curves fot eh respective classifiers given their tuned parameters. This is Shown below

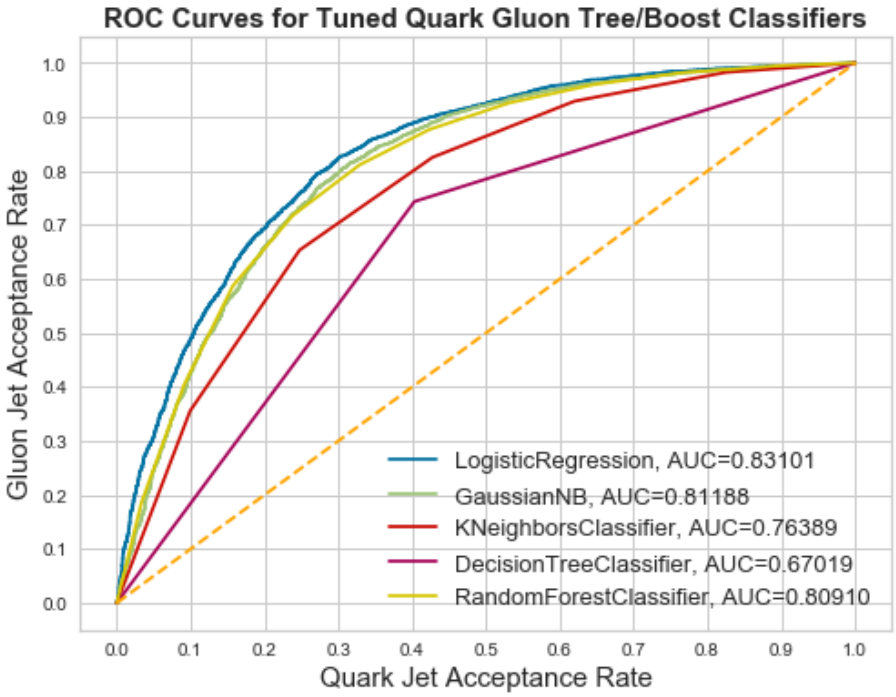


Figure 3.5: ROC curves for Untuned Classifiers

We can immediately see the improved AUC on the selected classifiers where the ROC curves are shown. Selecting the classifiers that produce the best results, the classification reports and class prediction errors are plotted bellow

The discrimination threshold (often set to 50 % by default) is the probability or score at which the positive class is chosen over the negative class. This can be calculated and plotted using the precision, recall, f1 score, and queue rate with respect to the discrimination threshold of a binary classifier. The discrimination threshholds are also plotted

3.6 Deep Learning

Loss tends to decrease as epochs go by, because our model is learning to minimize the loss function. After a certain amount of epochs, loss converges (reaches a minimum). Accuracy curves are essentially the opposite, the model becomes more accurate in its prediction as it is learning, or as epochs go by. If we plot training vs validation data, we can identify overfitting: when the model starts learning

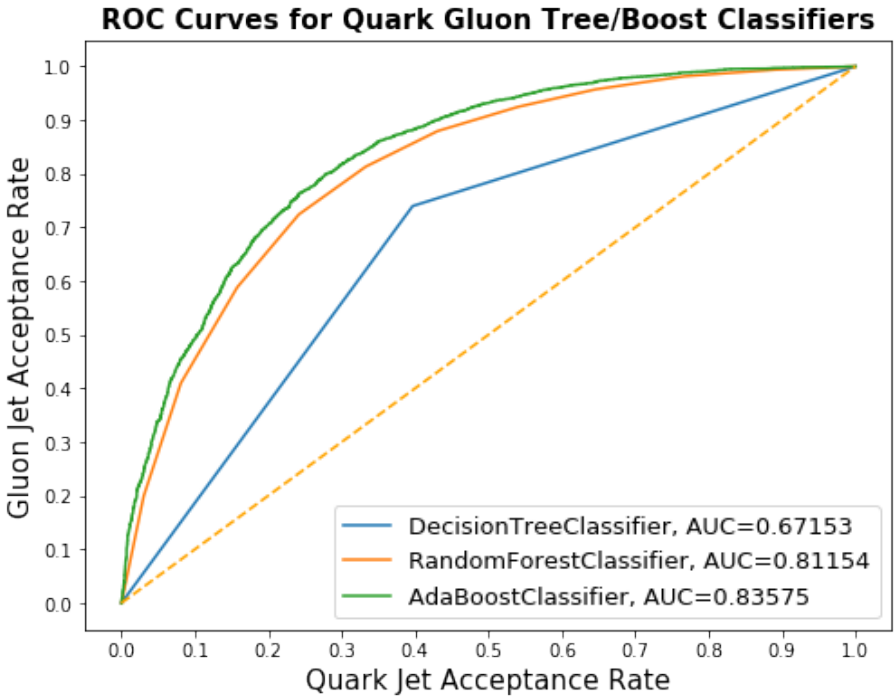
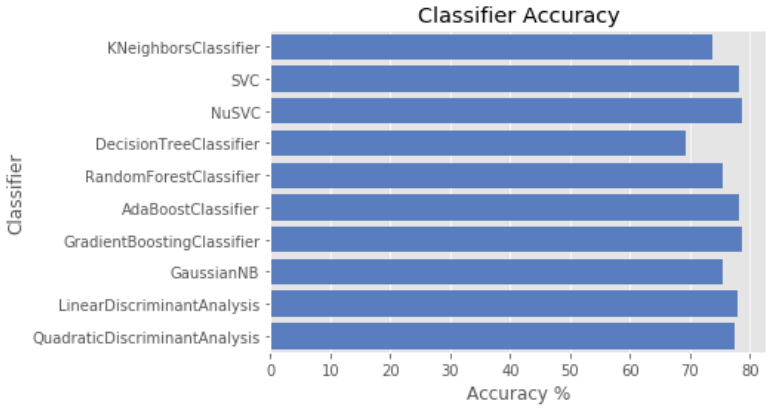


Figure 3.6: ROC Curves for Untuned Tree-based and Boosting Classifiers

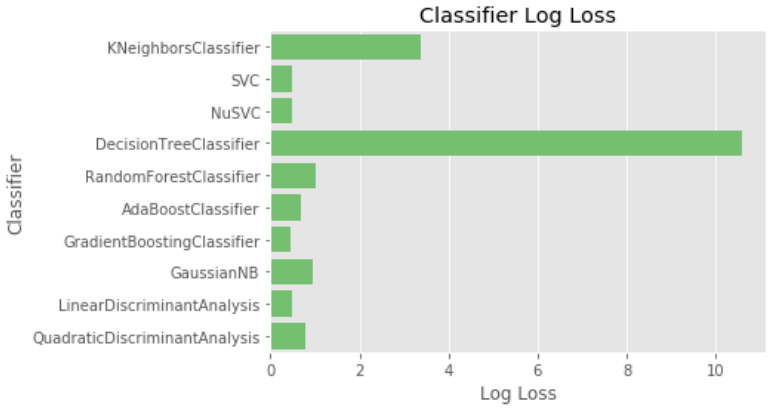
particularities of the trainig data which don't generalize well to test data. And example of this

A model that is underfit will have high training and high testing error while an overfit model will have extremely low training error but a high testing error. To see this consider the following loss and accuracy curves of my initial, overfit neural networks

After optimization of the neural network parameters, the optimal neural network was achieved. This neural network has testing and training errors that match each other as the neural network is training. The model summary for this optimal Neural Network is also provided



(a) Class Accuracy for Untuned Classifiers



(b) Log Loss for Untuned Classifiers

Figure 3.7: Accuracy and Log loss Classifier Comparison

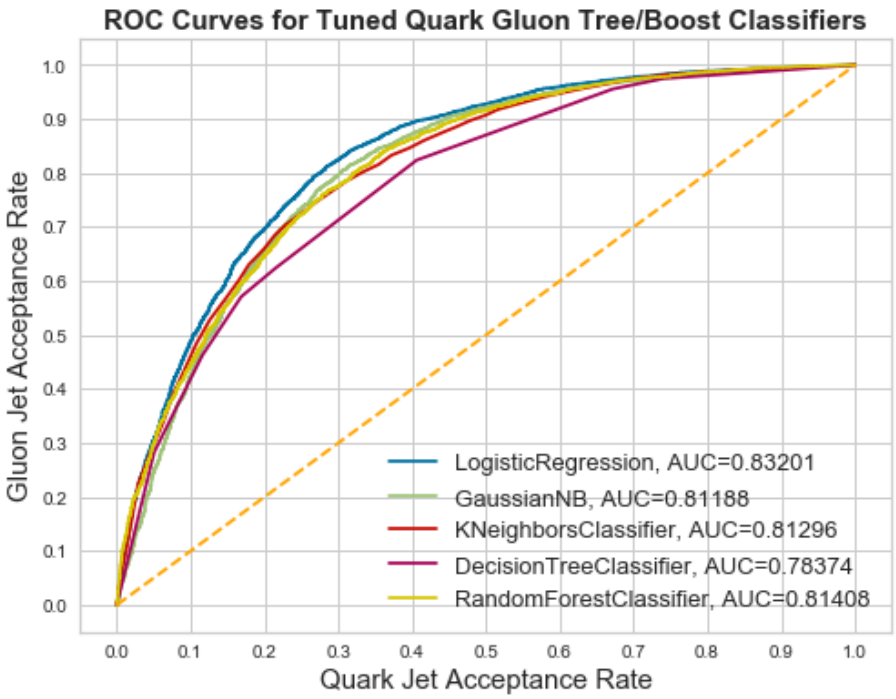
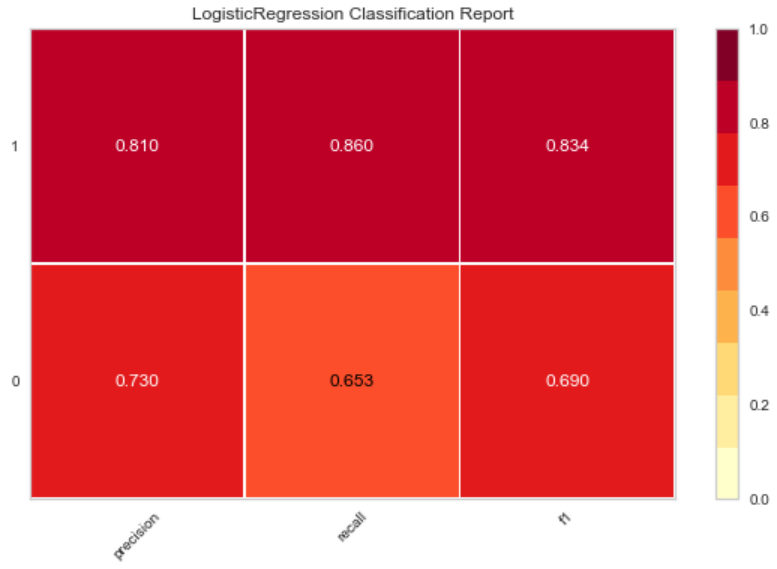
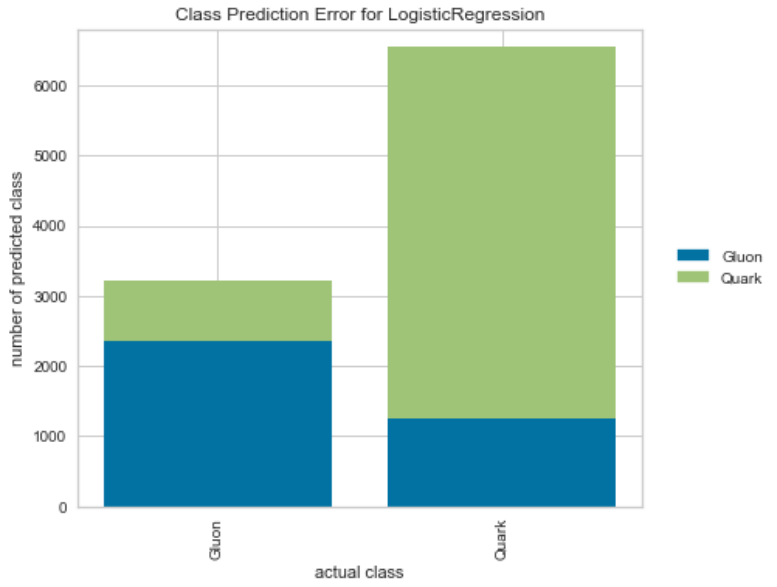


Figure 3.8: ROC Curves for Tuned Classifiers

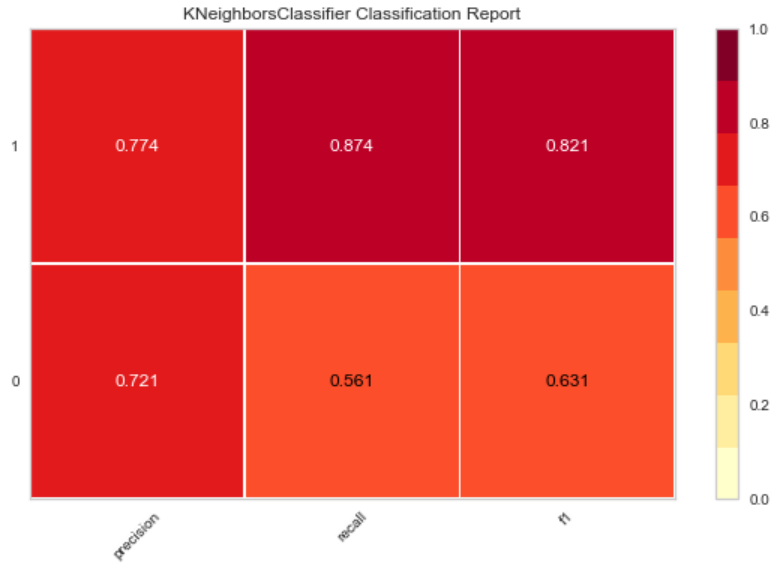


(a) Classification Report for Tuned Logistic Regressor

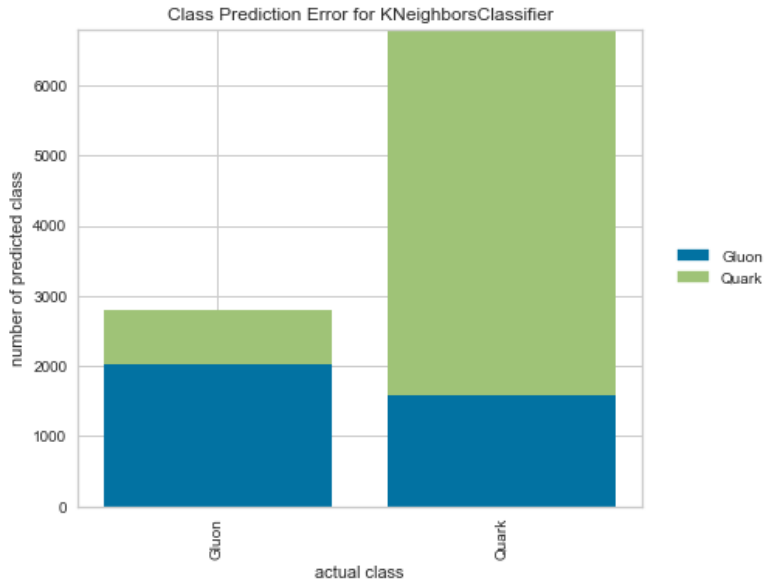


(b) Class Prediction Error for Logistic Regressor

Figure 3.9: Tuned Logistic Regression Classifier

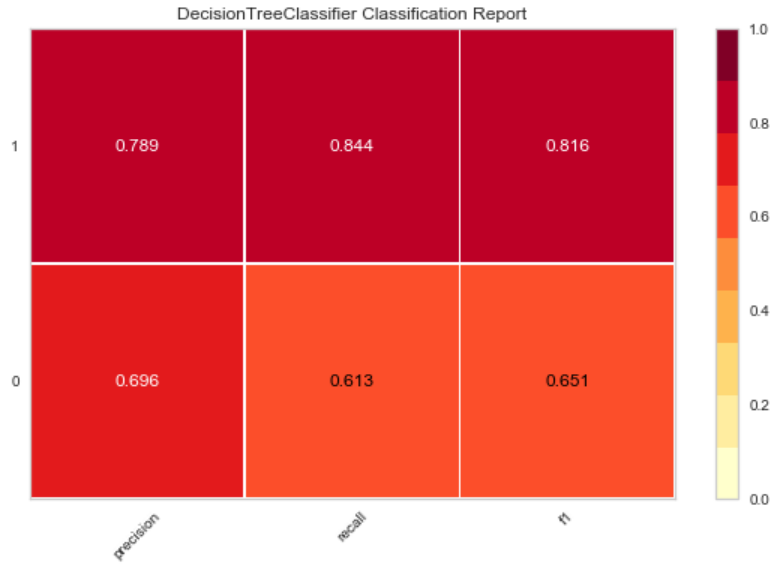


(a) Classification Report for Tuned K Nearest Neighbor



(b) Class Prediction Error for Tuned K Nearest Neighbor

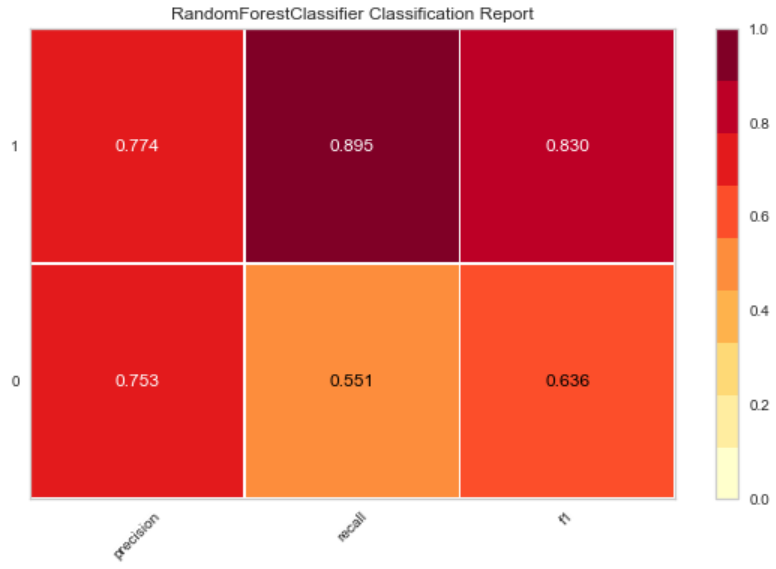
Figure 3.10: Tuned KNN Classifier



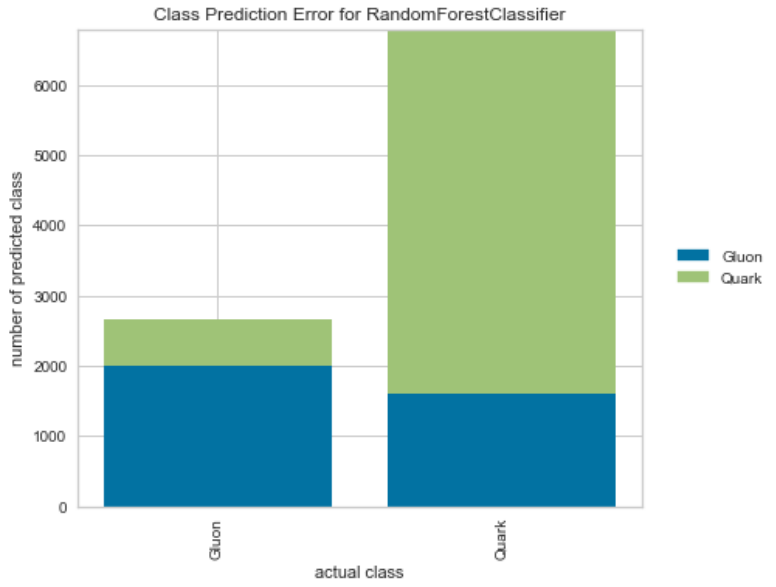
(a) Classification Report for Tuned Decision Tree

(b) Class Prediction Error for Decision Tree

Figure 3.11: Tuned Decision Tree Classifier



(a) Classification Report for Tuned Random Forest



(b) Class Prediction Error for Tuned Random Forest

Figure 3.12: Tuned Random Forest Classifier

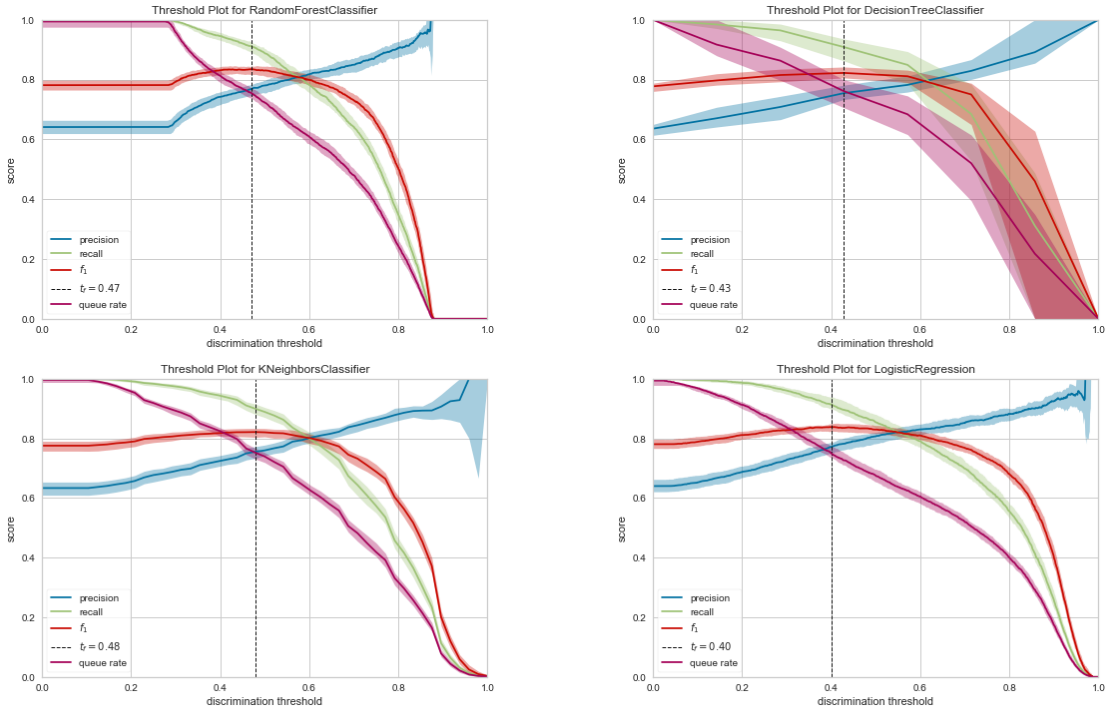
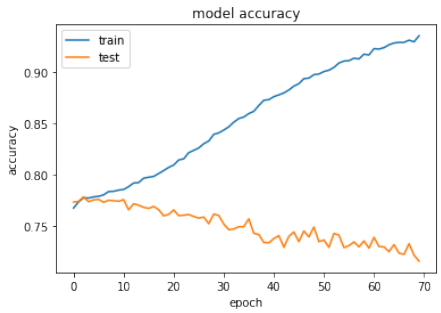
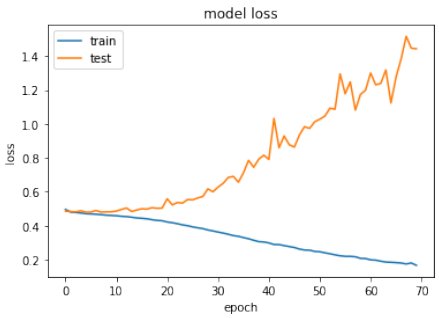


Figure 3.13: Threshhold Plots for Classifiers

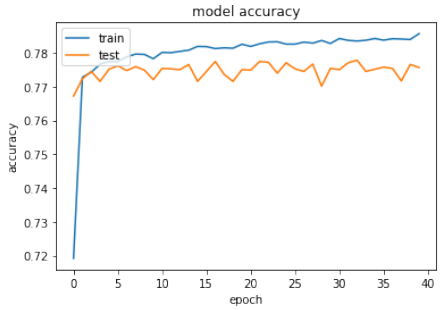


(a) Overfit Neural Network Model Accuracy

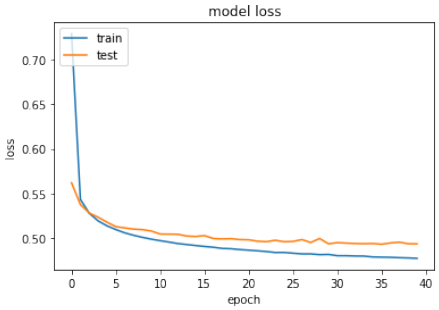


(b) Overfit Neural Network Model Loss

Figure 3.14: Overfit Neural Network



(a) Optimal Neural Network Model Accuracy



(b) Optimal Neural Network Model Loss

Figure 3.15: Optimal Neural Network

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 100)	1900
dense_5 (Dense)	(None, 200)	20200
dense_6 (Dense)	(None, 50)	10050
dense_7 (Dense)	(None, 200)	10200
dense_8 (Dense)	(None, 200)	40200
dense_9 (Dense)	(None, 1)	201

Total params: 82,751
Trainable params: 82,751
Non-trainable params: 0

Figure 3.16: Optimal Neural Network Model Summary

Chapter 4

Classifier Comparison to Contaminated Classifier

The objective of any of our classifiers is to approximated the function

$$\begin{aligned} f^* &= P(t=1|x) \\ &= \frac{p(x|s)\pi(s)}{p(x|s)\pi(s)+p(x|b)\pi(b)} \end{aligned} \tag{4.1}$$

In other words, the function f^* (or classifier) approximates the discriminant

$$D(x) = \frac{s(x)}{s(x)+b(x)} \tag{4.2}$$

However, this discriminant assumes ideal and pure probability densities $g(x)$ and $q(x)$. The actual densities of the samples have quark densities that include mixtures of the other quark flavor, and since we are only considering binary classification of quark and gluon jets, the actual densities are

$$\begin{aligned} G(x) &= (1-\epsilon_q)\, g(x) + \epsilon_q q(x) \\ Q(x) &= (1-\epsilon_g)\, q(x) + \epsilon_g g(x) \end{aligned} \tag{4.3}$$

Where the fractions ϵ_g and ϵ_q ¡1. Hence we can define the actual, or contaminated discriminant, which is what we are actually approximating by the classifiers, which includes the mixture fractions

$$D'(x) = \frac{G(x)}{G(x)+Q(x)} \tag{4.4}$$

Which, after subsitituting $G(x)$ and $Q(x)$ and simplifying

$$D'(x) = \frac{\epsilon_q q(x)_g(x)(1-\epsilon_q)}{\epsilon_g g(x) - \epsilon_g + \epsilon_q q(x) + g(x)(1-\epsilon_q) + 1} \tag{4.5}$$

Now, what is interesting is that $D(')(x)$ can be represented in terms of D. Substituting, we get

$$D'(D) = \frac{D\left(-2\epsilon_q + 1\right) + \epsilon_q}{D\left(2\epsilon_g - 2\epsilon_q\right) - \epsilon_g + \epsilon_q + 1} \tag{4.6}$$

This is very interesting as D' is now only a function of D, where ϵ_q and ϵ_g are constants ¡1. This means that the optimal discriminant D can be computed from D', where the contamination fractions are given in D', and by studying the relationship between D and D' over the range of ϵ_g and ϵ_q one could identify where the relationship between them is monotonic. The problem, interestingly, is even simpler, since $D'(D)$ is just a linear function of D. factoring out D, we get

$$D' = D\frac{(g+q)(eqq + g(1-eq))}{g(egg - eg + eqq + g(1-eq) + 1)} \tag{4.7}$$

Meaning the derivative dD'/dD for a random sampling of values of ϵ_g and ϵ_q is always positive, hence the mapping between D' and D is one-to-one

$$D' = \frac{(g+q)(eqq + g(1-eq))}{g(egg - eg + eqq + g(1-eq) + 1)} \tag{4.8}$$

Meaning that provided that the mapping between them is one-to-one, the discrimination power of D' is the same as that of the optimal discriminant D, provided that it has been accurately determined. Hence we can study the range of available classifiers, and study which of those map the best with D'.
Now, constructing this contaminated classified D' simply algebraicly from the studied classifiers, we can attain D' for a given classifier and for given values of $\{\epsilon_g, \epsilon_q\}$. Something interesting happens here: for a given classifier, if the classifier is not well tuned, then the ROC and AUC values for D and D' are not equal. However, if the classifier's parameters are well-tuned, then they are equal, for any given value of $\{\epsilon_g, \epsilon_q\}$ in the range $[0, 0.5]$. Let's take, for example, the ROC curve of an untuned Decision Tree Classifier.

Next, we construct D' for this particular classifier D for for a random sampling of $\{\epsilon_g, \epsilon_q\}$ in the range $[0, 0.5]$, and calculate the ROC curve and AUC for each

We note that the AUC values of this contaminated classifier are equal to the AUC of the original classifier for $\{\epsilon_g, \epsilon_q\} = \{.25, 0.15\}, \{0.5, 0.3\}, \{0.4, 0.4\}$. We also note (obviously!) that $\{\epsilon_g, \epsilon_q\} = \{0.5, 0.5\}$, AUC =0.5 since they have equal probabilities.
However, if we plot the tuned (Decision Tree) classifier D, we have the following

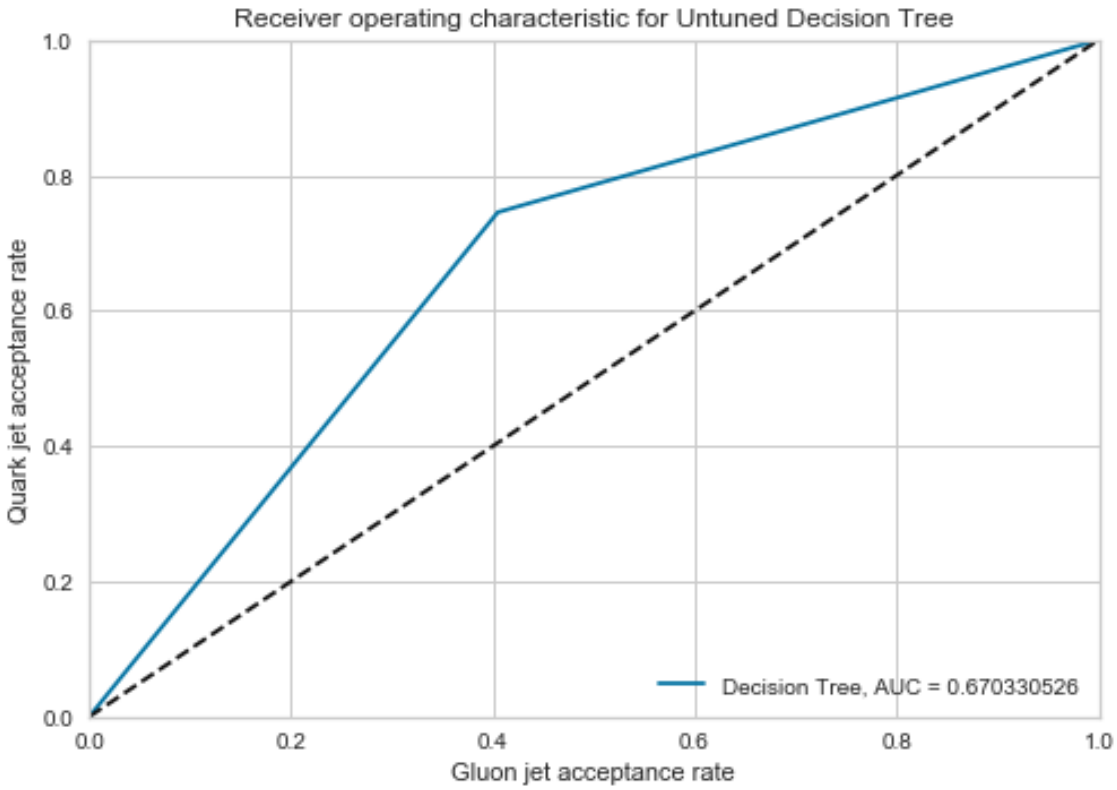


Figure 4.1: ROC Curve for Untuned Decision Tree

ROC and AUC: And the corresponding ROC curves for D' We see that now, most of the combinations of $\{\epsilon_g, \epsilon_q\}$ agree with their AUC (and ROC shape) to the classifier D. A similar situation happens with all the studied classifiers.
Let's study the behavior of my optimized neural network, since it has given very good accuracy compared to the other classifiers. The Neural Network (described in Chapter 3) has the following ROC curve.

And applying this classifier to the data, we can see the resulting quark-gluon jet discriminant histogram

And we can construct D' from this D for various values of $\{\epsilon_g, \epsilon_q\}$:

We note that the ROC and AUC values for the contaminated classifier is the same as that of the original classifier D for all the considered values of the mixtures $\{\epsilon_g, \epsilon_q\}$ in the range $[0, 0.5]$.
Furthermore, we can plot D vs D' for various values of $\{\epsilon_g, \epsilon_q\}$ and see whether they are D is monotonic in D' :

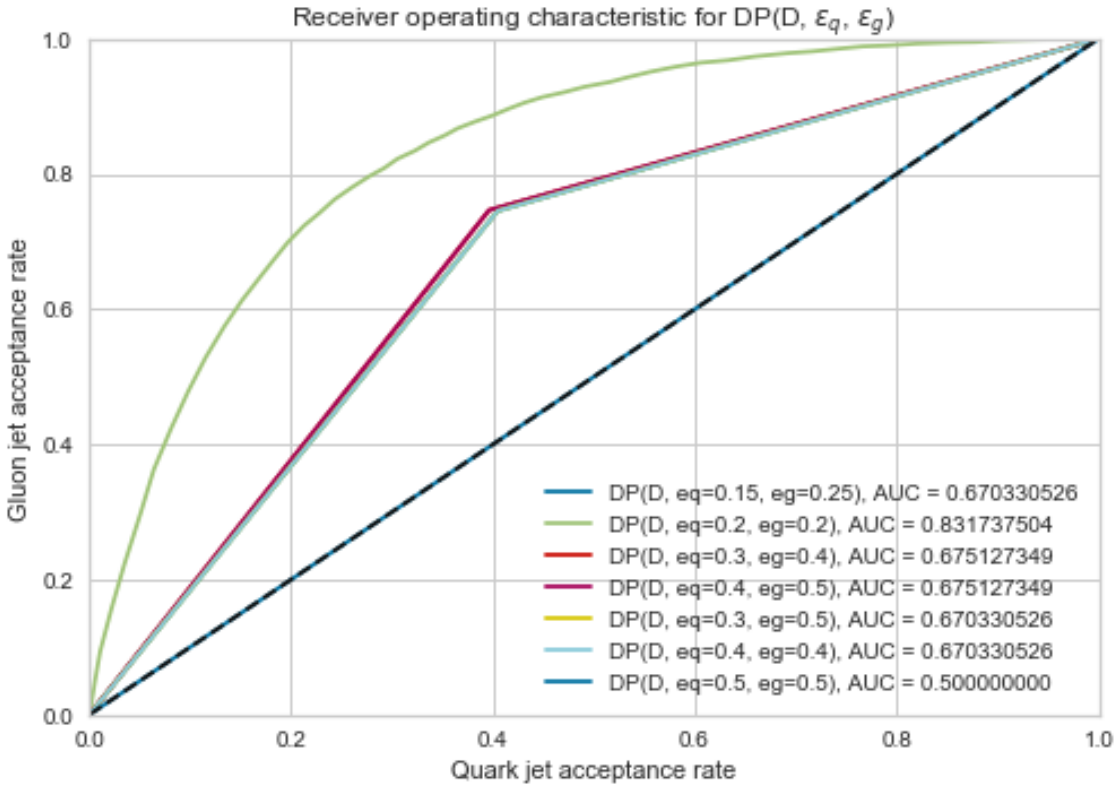


Figure 4.2: D' ROC Curves for Untuned Decision Tree

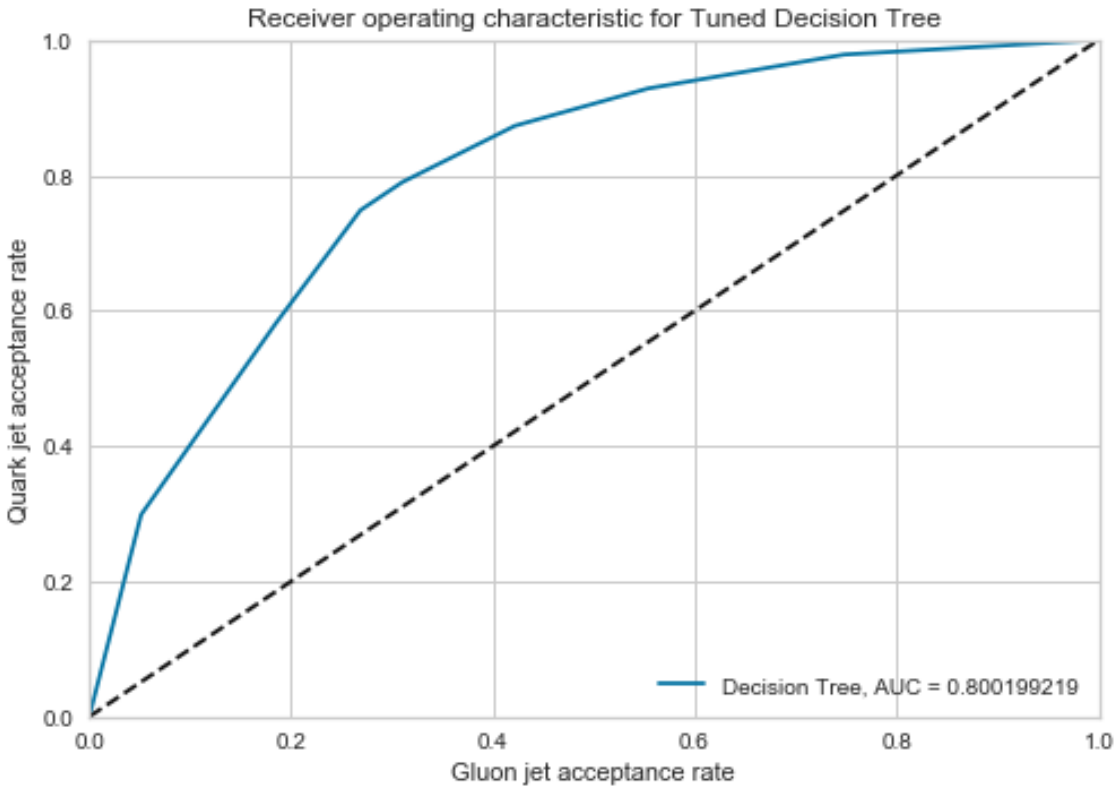


Figure 4.3: ROC Curve for Tuned Decision Tree D

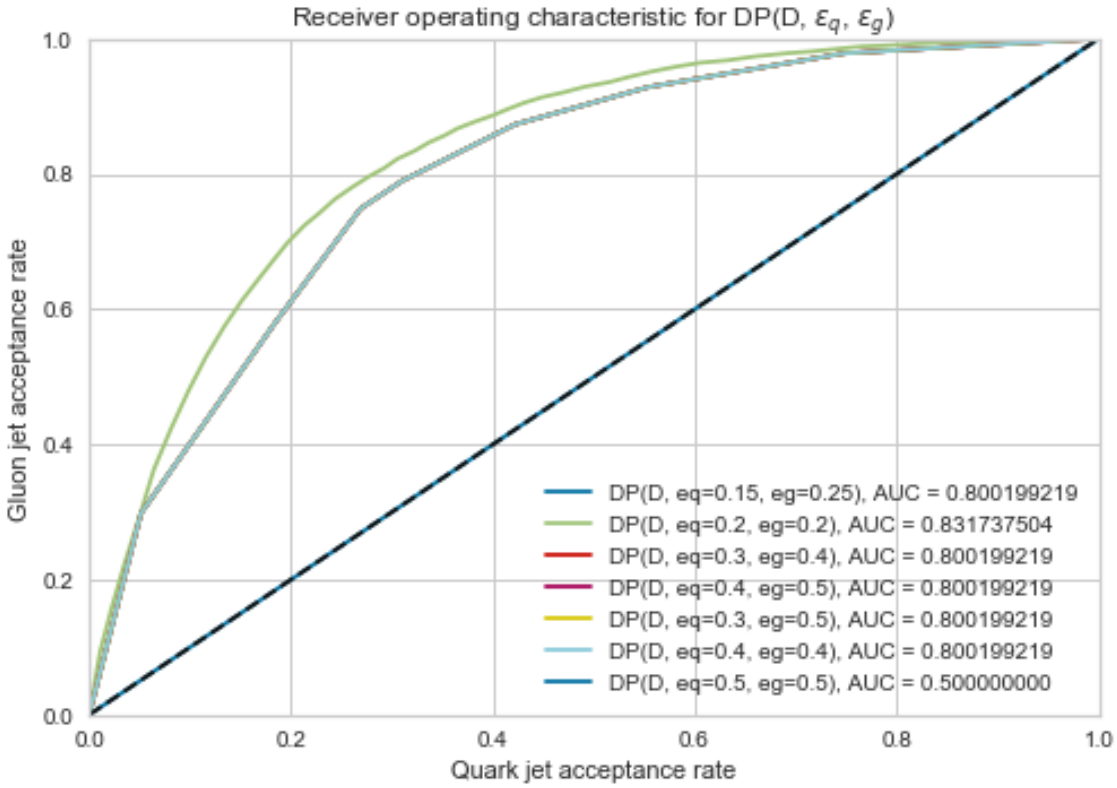


Figure 4.4: ROC Curve for D' corresponding to Tuned Decision Tree

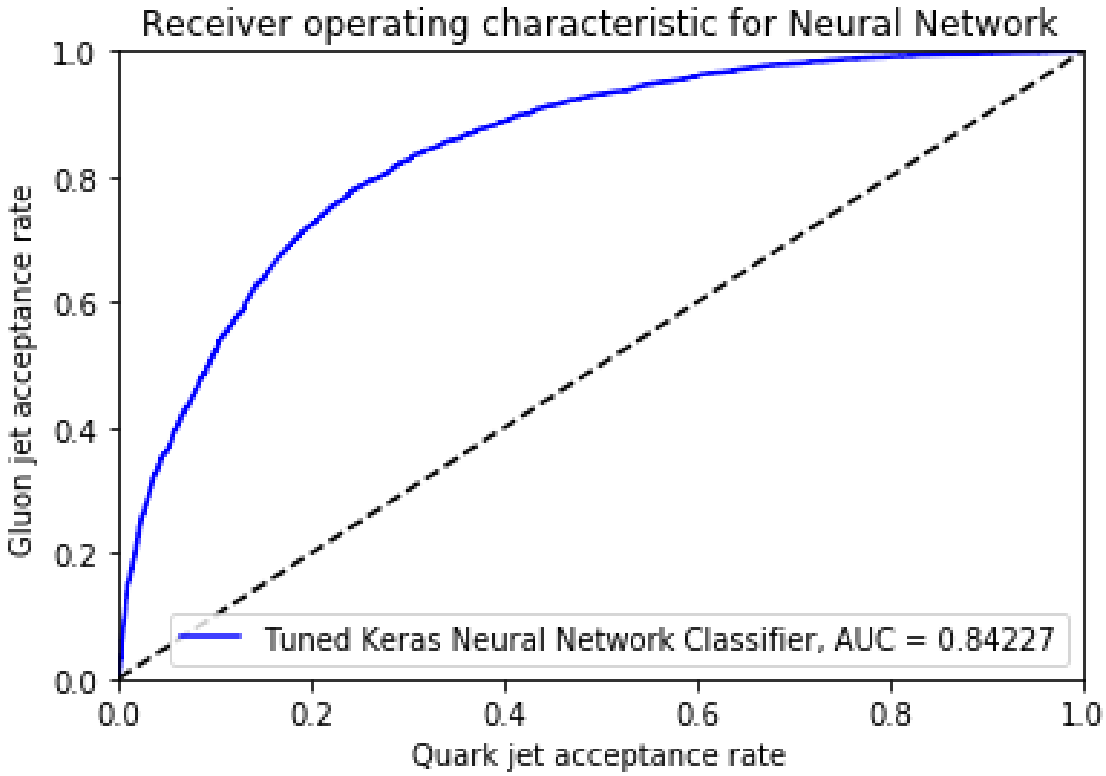


Figure 4.5: ROC Curve for Optimized Neural Network D

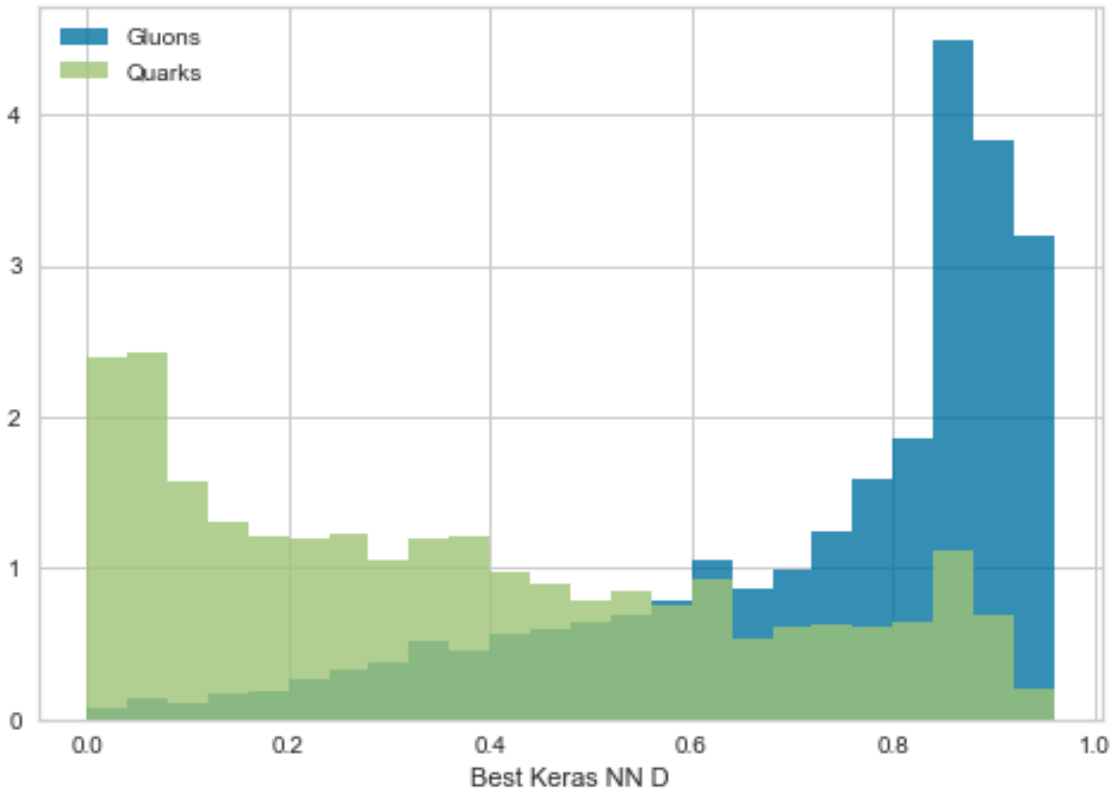


Figure 4.6: Quark-gluon jet distributions using Neural Network D

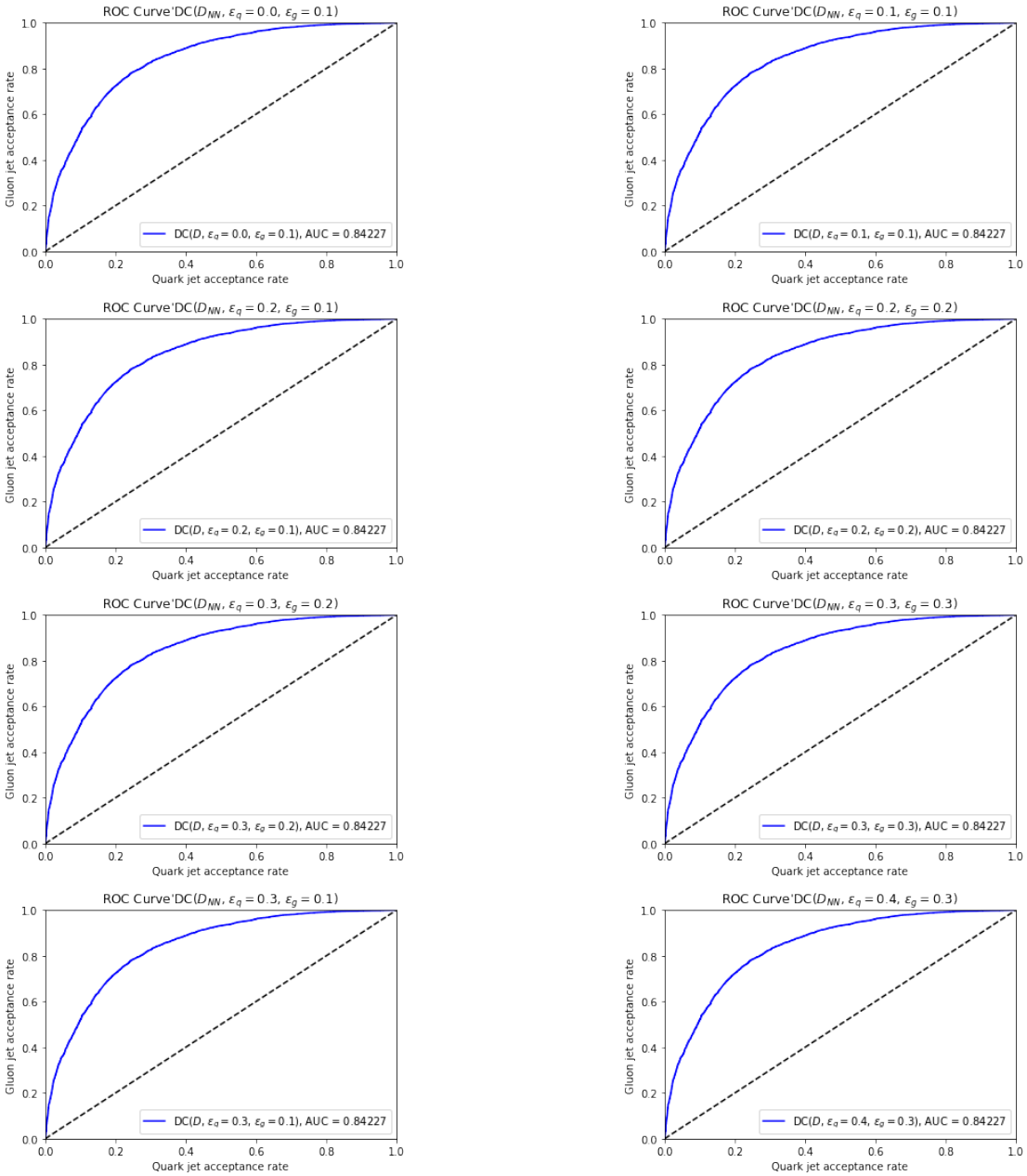


Figure 4.7: Contaminated Classifier D' Corresponding to Optimized Neural Network, evaluated at various values of $\{\epsilon_g, \epsilon_q\}$

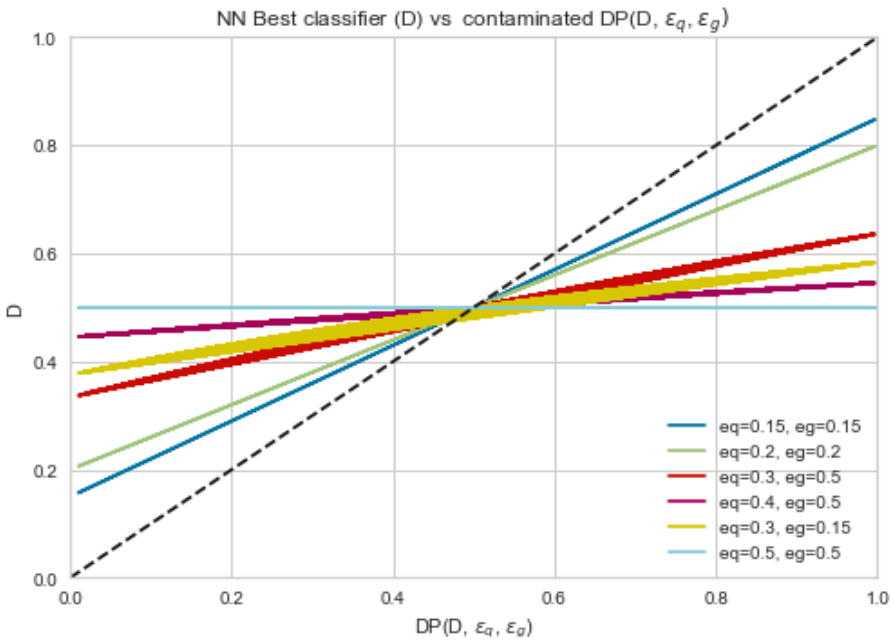


Figure 4.8: D vs D'

Chapter 5

Conclusion and Outlook

Appendix A

Appendix Title

Contents

Bibliography

[1] Kimmo Kallonen. *Sample with jet properties for jet-flavor and other jet-related ML studies JetNTuple QCD RunII 13TeV MC*. URL: http://opendata.cern.ch/record/12100.