Classifying Cherenkov Radiation
The MAGIC Telescope

Peter Trubey

May 11, 2013

Abstract

The Purpose of this paper is to demonstrate competing methods for classification. The methods we will show include using a single decision tree from the *rpart* package, both unweighted and weighted to punish false positives more than false negatives. We will also use random forests from the *randomForest* package, and a novel method using bootstrap aggregated neural networks. For this we will use the *nnet* package, and write our own wrapper to handle the bootstrap sampling, model building, and voting.

1 Introduction

Gamma radiation refers to electromagnetic radiation with a high frequency, and thus extremely high energy per photon. Gamma rays have a shorter wavelength, and therefore higher frequency than x-rays, and have much higher energy. Where x-rays may have energies of tens of eV, this paper is about detecting radiation from 30 to 400 GeV [4] [5]. Though there exist some terrestrial sources of gamma radiation, such as through gamma decay of radioisotopes, this paper focuses on methods of detecting gamma rays of an extraterrestrial origin, which are produced through astronomical processes. These might include such events as active galactic nuclei, supernova remnants, gamma ray bursts, and other sources not yet identified [3].

A gamma ray telescope attempts to observe gamma rays emitted as a result of these celestial events. Due to the fact most gamma rays are filtered out by the atmosphere, typically we will attempt to detect these gamma rays from a satellite, orbiting the earth. By contrast, the MAGIC (Major Atmospheric Gamma-ray Imaging Cherenkov) telescope this data is supposed to represent is terrestrially mounted, and observes the effects of high energy gamma photons colliding with the atmosphere. When these gamma photons, travelling at light speed in space interact with the atmosphere, they collide with a medium where the speed of light is slower than it was in space. This releases Cherenkov photons, which the MAGIC gamma telescope can then observe. Telescopes that work in this manner are referred to as Imaging Atmospheric Cherenkov Telescopes (IACT's). Depending on the energy of the primary gamma, some number of Cherenkov photons are collected, and through interaction with a photomultiplier tube [4] an image of the shower can be created. It is from this shower image that the descriptive attributes are collected. We attempt

to classify whether a given shower was caused by a primary gamma ray, or hadronic showers from cosmic rays.

2 Data

This data does not originate from actual observations. It is meant to represent these events, ranging from 50 to 300 GeV. It was computer generated using a Monte Carlo program called CORSIKA [1]. The dataset contains 19020 observations, composed of 10 continuous attributes describing the shower image, and the classification.

Table 1: Variable Descriptions [2]

Variable	Type	Description
fLength	Continuous	Major Axis of Ellipse (mm)
fWidth	Continuous	Minor Axis of Ellipse (mm)
fSize	Continuous	Base 10 Log of sum of content of all pixels (# of Photons Collected)
FConc	Continuous	Ratio of sum of to highest pixels over fSize
fConcl	Continuous	Ratio of highest pixel over fSize
fAsym	Continuous	Distance from Highest Pixel to Center, Projected onto Major Axis (mm)
fM3Long	Continuous	Third root of third moment along major axis (mm)
fM3Trans	Continuous	Third root of third moment along minor axis (mm)
fAlpha	Continuous	Angle of Major axis with vector to origin (degrees)
fDist	Continuous	Distance from origin to center of ellipse (mm)
class	1,0	Gamma (1, Signal), Hadron (0, Background)

These are all metrics gathered from the pattern of the ellipse after some pre-processing to remove noise [3]. As we can see from their histograms in figure 3, fLength and fWidth are heavily skewed. Owing to the nature of the methods I will be using, I will be performing transformations on the predictor variables to limit this skew. To that end, we will be taking the square root of fLength and fWidth. I will also be scaling all attributes to have a mean of 0, and a variance of 1. These steps are necessary for fitting an artificial neural network, though it should have no effect on the other methods used.

Also in looking at the histograms (figures 3, 4, 5), we can see how closely the gamma and hadron ray diagnostics resemble each other in terms of distributions. The only variable that shows a large difference is fAlpha, in which the distributions are wildly different. The density for the gamma appears to be exponentially decreasing, where the density for the hadron appears uniform.

3 Methods

The methods we will use include a single decision tree using the CART algorithm in the package *rpart*, both unweighted and weighted to punish false classification of a hadron as a gamma ten times greater than the opposite error. In pruning these trees, we will be using ten-fold cross-validation to find a minimum error, and then applying the one standard error rule. We will also be implementing a random forest using the package *randomForest*. As comparison to these tree based methods, we shall also implement a bootstrap aggregated neural network, with voting power based on the out of bag misclassification rate. The package used to build the neural networks is *nnet*, but I will be writing a custom wrapper following a modified bagging algorithm to implement the bootstrap sampling and vote aggregating.

Algorithm 1: Bagged Artificial Neural Network
With Weighted Voting

For Building Ensemble with B Classifiers

Pass Training set;
for *i in 1 to B* **do**
    Generate Bootstrap Sample of Size n;
    Train Neural Network based on Bootstrap Sample, Random Weights;
    Test Neural network performance on *Out of Bag* Sample;
    Record performance;
end

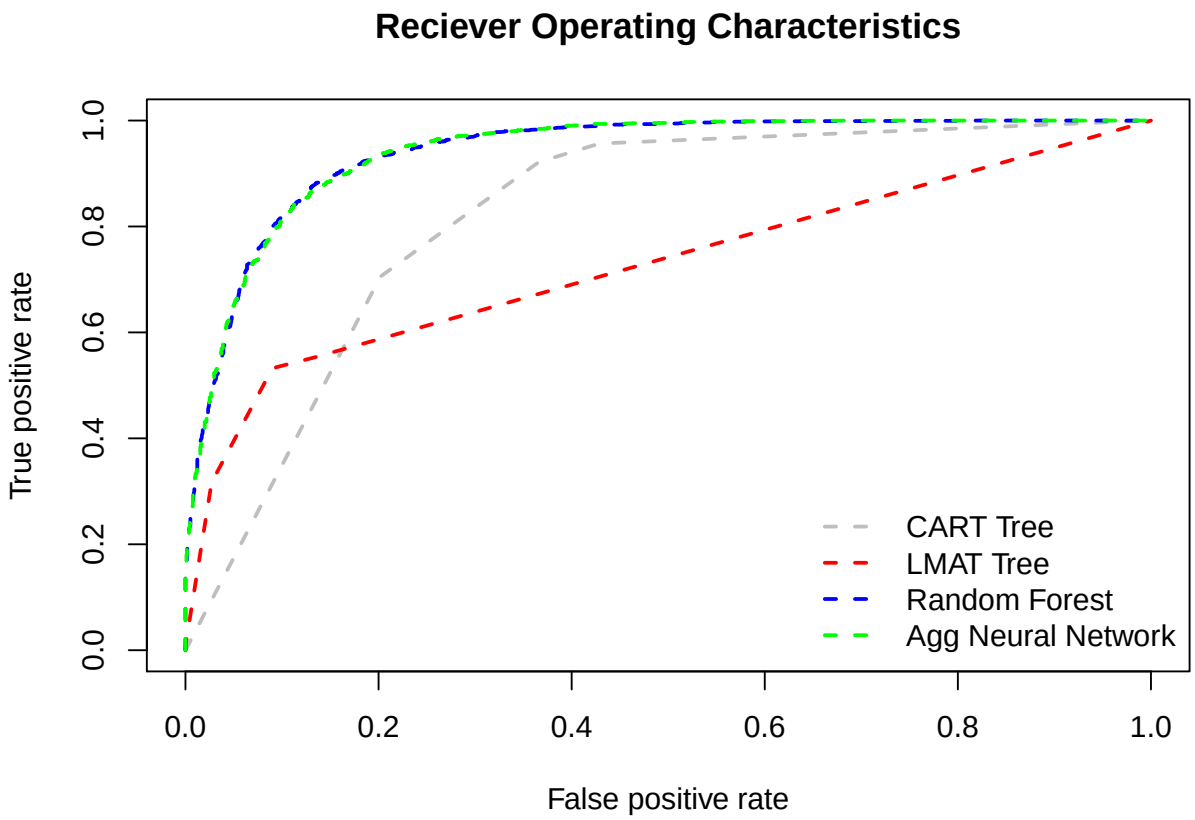
For obtaining Class Probability Predictions from Ensemble

Pass Test Set;
for *i in 1 to B* **do**
    Predict Test Set classification using Neural Network i;
    Record P(Observation Class = 1) in Matrix;
end
Weights = (OOB performance - min(OOB Performance))^2;
Weights = Weights / Sum(Weights);
Return [Weights] X [Prediction Matrix];

4 Analysis and Results

As we can see from the ROC Curve in figure 1, the ensemble methods perform much better than a single tree. Unfortunately, the ensemble methods are so close to each other, that we cannot say that one method is definitively better than another. We look to the table of true positive rates in table 2 to decide which is better at a given threshold. Therefore we offer the aggregate neural network as an alternative to random forest, since the results appear to be comparable for

Figure 1: ROC Curve Comparing Trees to Aggregated
Neural Networks, Random Forest



this dataset. The false positive thresholds given in the table were suggested by the contributor of the dataset [2] as a means to compare classification algorithms. When we observe the variable importance plots in figure 2, we see what we expected in looking at the histograms: fAlpha dominated the other variables in importance. We expected this because fAlpha is the only variable that featured significantly different density plots between the gamma and hadron classifications.

5 Conclusion

We had hoped that one method might emerge a clear victor by yielding a much stronger result than the others. What we observed is that both the ensemble methods we tried yielded a similar result. It may be that on a different data set, the aggregated neural network will perform better than random forest. It is clear that both are comparable here. It may also be possible that the aggregated neural network will work better here, as we may not have tried the best transformation of the data. While decision trees (and therefore random forest) are not sensitive to monotone transformations, neural networks definitely are. Thus there may be something we haven't tried yet that would yield a better result.

Table 2: True Positive Rates and Class Probability
Cutoffs at False Positive Thresholds

FP Rate	.01	.02	.05	0.10	0.20
Modified Loss	0.0084	0.0084	0.3166	0.5341	0.5529
Tree	1	1	0.9653	0.6984	0.5359
Random Forest	0.3122	0.4301	0.6356	0.817	0.9301
	0.959	0.929	0.857	0.734	0.523
Aggregated	0.3253	0.4306	0.6503	0.8138	0.9341
Neural Network	0.9593	0.9344	0.8642	0.7555	0.537

True Positives Class Probabilities

References

[1] CORSIKA. http://www-ik.fzk.de/~corsika/.

[2] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[3] Wagner Robert. MAGIC - the MAGIC telescope, 2009.

[4] R.T. Weidner and R.L. Sells. *Elementary Modern Physics*. College physics series. Allyn and Bacon, 1968.

[5] H.D. Young. *Fundamentals of optics and modern physics*. McGraw-Hill series in fundamentals of physics. McGraw-Hill, 1968.

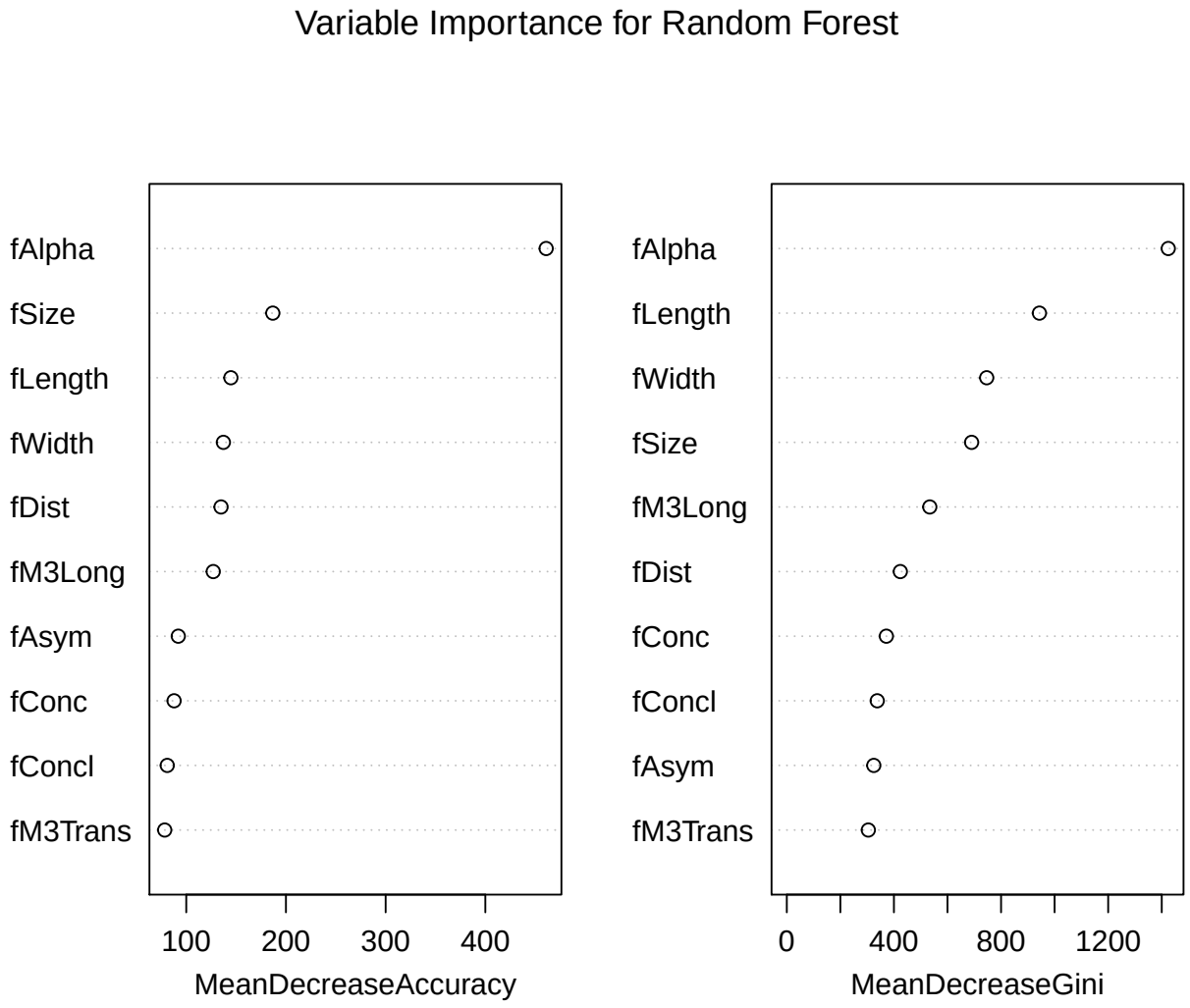


Figure 2: The Variable Importance Plots from the Random Forest Classifier

Figure 3: Distributions of Predictors relative to Class [1]

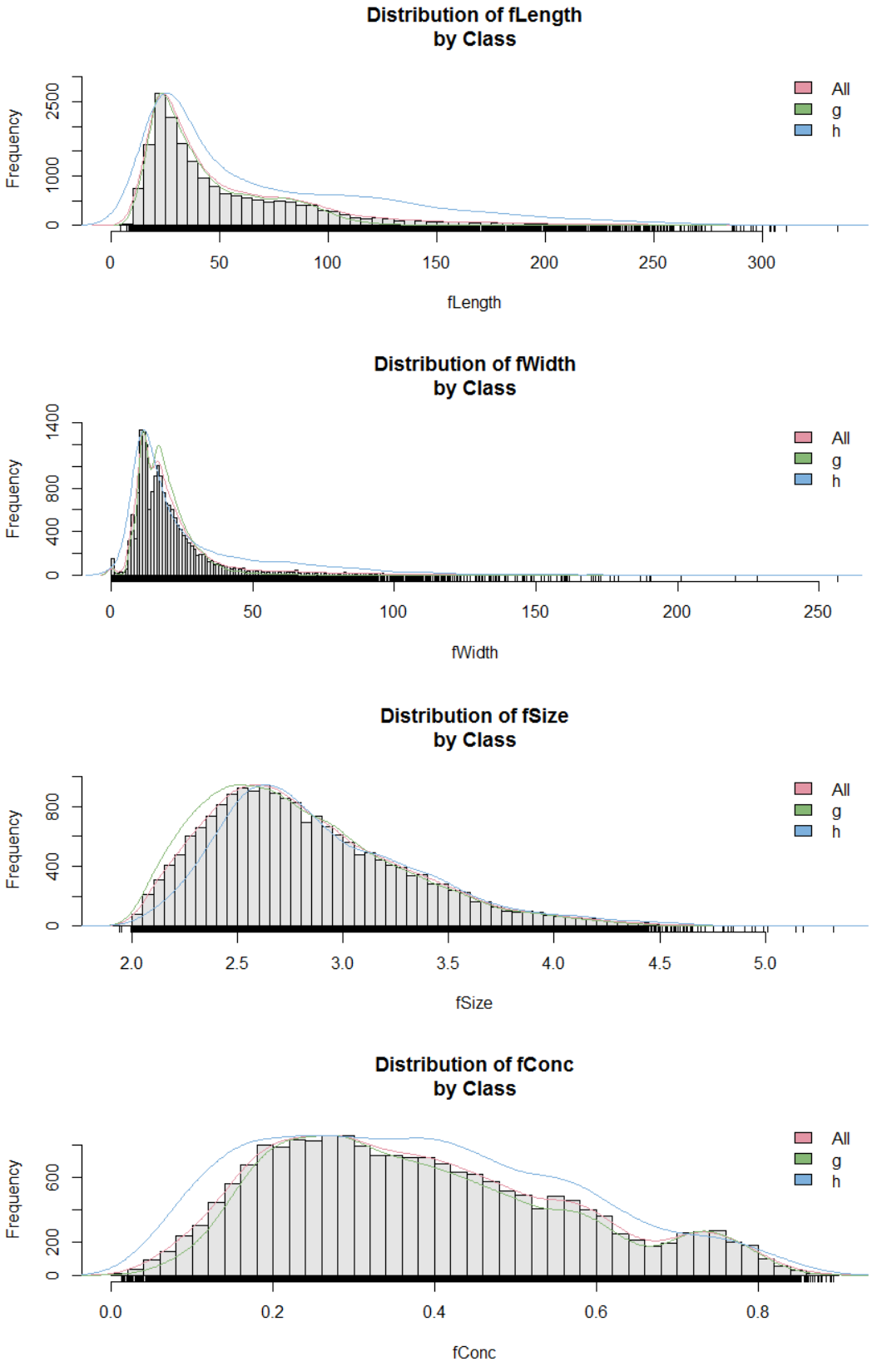


Figure 4: distributions of Predictors relative to Class [2]

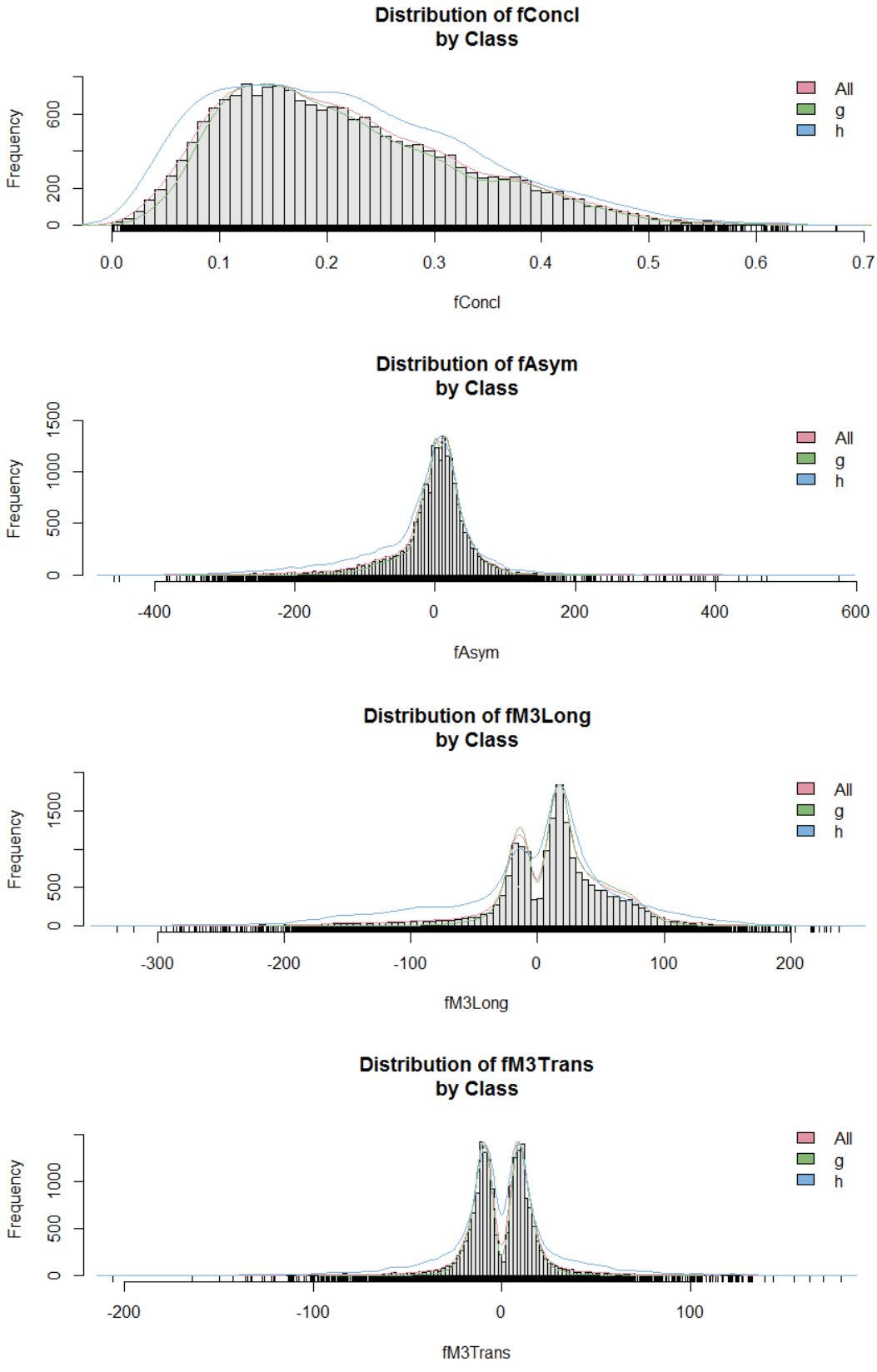
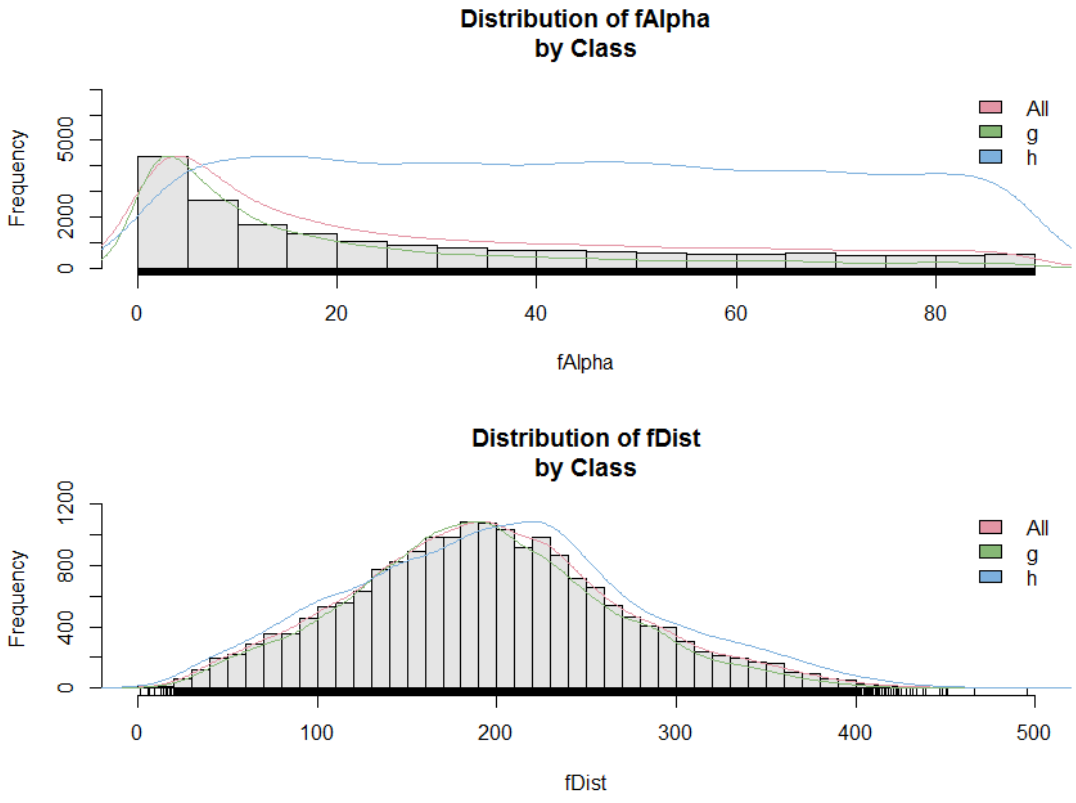


Figure 5: Distributions of Predictors relative to Class [3]



Code Block 1: Code Used in Document

```
source("dataprep.R")
source("annet.R")

# Building Rpart Tree Model
lmat = matrix(c(0, 1, 10, 0), nrow = 2)
gamma.tree = rpart(class ~ ., data = data$train, method = "class")
gamma.tree = prune(gamma.tree, cp = 0.0107)
gamma.tree.lmat = rpart(class ~ ., data = data$train, method = "class",
    parms = list(loss = lmat))
gamma.tree.lmat = prune(gamma.tree.lmat, cp = 0.011)
gamma.tree.pred = prediction(predict(gamma.tree, newdata = data$val, type = "prob")[,
    2], data$val$class)
gamma.tree.lmat.pred = prediction(predict(gamma.tree.lmat, newdata = data$val,
    type = "prob")[, 2], data$val$class)
plot(performance(gamma.tree.pred, "tpr", "fpr"), col = "gray", lty = 2,
    lwd = 2, main = "Reciever Operating Characteristics")
plot(performance(gamma.tree.lmat.pred, "tpr", "fpr"), col = "brown", lty = 2,
    lwd = 2, add = T)

# Building Random Forest Model

gamma.rf <- randomForest(class ~ ., data = data$train, mtry = 3, ntree = 1000,
    keep.forest = TRUE, importance = TRUE)
gamma.rf.pred = prediction(predict(gamma.rf, newdata = data$val, type = "prob")[,
    2], data$val$class)
plot(performance(gamma.rf.pred, "tpr", "fpr"), lty = 2, lwd = 2, col = "pink",
    add = T)

# Building Bagged Neural Network

gamma.annet = construct.annet(data$train, 50)
gamma.annet.pred = prediction(predict.annet(gamma.annet, data$val, 2),
    data$val$class)
plot(performance(gamma.annet.pred, "tpr", "fpr"), lty = 2, lwd = 2, col = "green",
    add = T)

gamma.tree.lmat.perf = thresholds(performance(gamma.tree.lmat.pred, "tpr",
    "fpr"))
gamma.rf.perf = thresholds(performance(gamma.rf.pred, "tpr", "fpr"))
gamma.annet.perf = thresholds(performance(gamma.annet.pred, "tpr", "fpr"))
```

Code Block 2: Data Preparation

```
dataprep = function() {
    data = read.table("magic04.data", sep = ",", header = F, col.names = c("fLength",
        "fWidth", "fSize", "fConc", "fConcl", "fAsym", "fM3Long", "fM3Trans",
        "fAlpha", "fDist", "class"), fill = FALSE, strip.white = T)

    # reordering the data - May not be necessary

    data$class = factor(ifelse(data$class == "g", 1, 0))
    shuffle = sample(dim(data)[1])
    data = data[shuffle, ]

    # normalizing the inputs

    data$fLength = scale(sqrt(data$fLength), center = T, scale = T)
    data$fWidth = scale(sqrt(data$fWidth), center = T, scale = T)
    data$fSize = scale(data$fSize, center = T, scale = T)
    data$fConc = scale(data$fConc, center = T, scale = T)
    data$fConcl = scale(data$fConcl, center = T, scale = T)
    data$fAsym = scale(data$fAsym, center = T, scale = T)
    data$fM3Long = scale(data$fM3Long, center = T, scale = T)
    data$fM3Trans = scale(data$fM3Trans, center = T, scale = T)
    data$fAlpha = scale(data$fAlpha, center = T, scale = T)
    data$fDist = scale(data$fDist, center = T, scale = T)

    # splitting the data into training and validation sets

    sample = rbinom(dim(data)[1], 1, 0.3)
    trainset = data[sample == 0, ]
    valset = data[sample == 1, ]

    return(list(train = trainset, val = valset))
}
data = dataprep()
library(rpart)
library(nnet)
library(randomForest)
library(ROCR)
```

Code Block 3: Aggregate Neural Networks

```
construct.annet = function(trainset, n, p = 1) {
    annet.struct = vector(mode = "list", length = n)
    pred = vector(mode = "list", length = n)
    cat("Constructing Agregate Neural Network with ", p, "\n")
    for (i in 1:n) {
        cat("iteration ", i, "\n")
        ibsamp = sample(c(1:nrow(trainset)), replace = T)
        annet.struct[[i]] = nnet(class ~ ., data = trainset[ibsamp, ],
            size = sample(4:12, 1), maxit = 2000)
        obsamp = setdiff(c(1:nrow(trainset)), ibsamp)
        pred.table = table(predict(annet.struct[[i]], newdata = trainset[obsamp,
            ], type = "class"), trainset[obsamp, ]$class)
        annet.struct[[i]]$perf = (pred.table[1, 1] + pred.table[2, 2])/length(obsamp)
        cat("Performance", annet.struct[[i]]$perf, "\n")
    }
    return(annet.struct)
}

predict.annet = function(annet.struct, testset, j = 2) {
    predictions = matrix(rep(NA, dim(testset)[1] * length(annet.struct)),
        ncol = dim(testset)[1])
    weights = c(rep(0, length(annet.struct)))
    for (i in 1:length(annet.struct)) {
        predictions[i, ] = predict(annet.struct[[i]], testset, type = "raw")
        weights[i] = annet.struct[[i]]$perf
    }
    weights = (weights - min(weights))^j
    weights = weights/sum(weights)
    preds = weights %*% predictions
    return(as.vector(preds))
}

thresholds = function(perf) {
    cutoffs = data.frame(cut = perf@alpha.values[[1]], fpr = perf@x.values[[1]],
        tpr = perf@y.values[[1]])
    cutoffs = cutoffs[order(cutoffs$tpr, decreasing = TRUE), ]
    thresholds = c(0.01, 0.02, 0.05, 0.1, 0.2)
    dat = matrix(c(rep(0, 10)), ncol = 5)
    for (i in 1:length(thresholds)) {
        dat[1, i] = round(head(subset(cutoffs, fpr < thresholds[i]))[1,
            3], 5)
        dat[2, i] = round(head(subset(cutoffs, fpr < thresholds[i]))[1,
            1], 5)
    }
    return(rbind(thresholds, dat))
}
```