STAT 27850 1 Multiple Testing, Modern Inference, and Replicability Project 2

Tytus Wilam, Simone Zhang, Aim Wonghirundacha, Soren Dunn

TOTAL POINTS

78 / 100

QUESTION 1

1 Project 2 **78 / 100**

Questions, ideas, & design of analysis (40pts) [Data option only]

* The project is designed in a creative and thoughtful way, to address interesting questions and challenges in the data

* The real world meaning of the data is considered in an insightful way to guide the design of the analysis

* Choices made along the way, for example designing a test statistic or finding a way to measure or visualize results, are addressed thoughtfully

* The analysis shows thorough understanding of any preexisting tools, code, packages, etc, that the group chose to use, and these choices are well suited to the problem at hand

+ 37 pts Click here to replace this description.

✓ **+ 34 pts** *Click here to replace this description.*

Statistical methodology (30pts) [Data option only]

* The analysis identifies all the major relevant challenges, such as issues of multiple testing, non-independence, confounding, exploratory data analysis, etc

* These challenges are handled appropriately in the analysis, applying existing tools or developing new techniques

* Assumptions are tested and examined using, e.g., using diagnostic plots

* Any remaining issues that cannot be addressed, are discussed

+ 26 pts Click here to replace this description.

✓ **+ 22 pts** *Click here to replace this description.*

Statistical methods / ideas / simulation design (70pts) [Theory option only]

* The project offers an insightful overview of the paper, with a well presented summary and with thoughtful comments and/or critique on the paper

* The report poses interesting and deep questions regarding extensions or limitations of the paper

* (If applicable) The new methods proposed in the project are well motivated and

thoughtfully constructed
* (If applicable) The new theoretical results developed in the project are correct, are clearly presented and proved, and are justified in terms of why they are meaningful to the methodology
* (If applicable) The simulations or experiments in the project are designed in a thoughtful and interesting way to address important questions about the properties or limitations of the methods being tested, and their results are recorded and interpreted well
* Any remaining issues that cannot be addressed, are discussed

+ 66 pts Click here to replace this description.

+ 56 pts Click here to replace this description.

+ 64 pts Click here to replace this description.

+ 55 pts Click here to replace this description.

+ 65 pts Click here to replace this description.

+ 50 pts Click here to replace this description.

+ 58 pts Click here to replace this description.

+ 54 pts Click here to replace this description.

Report & code (30pts)
* The report is clear and well-written, presenting a cohesive and well motivated explanation of the path followed in the analysis, and thoughtful and justified conclusions based on the findings
* Open questions, uncertainties due to insufficient data, questions relating to untestable assumptions, etc, are addressed as needed
* The code is clear, well organized, and appears readable and reproducible
* Sufficient details are given to understand the specifics of the analysis being run and the choices made along the way

+ 27 pts Click here to replace this description.

✓ **+ 22 pts** *Click here to replace this description.*

+ 29 pts Click here to replace this description.

+ 24 pts Click here to replace this description.

+ 25 pts Click here to replace this description.

+ 28 pts Click here to replace this description.

+ 26 pts Click here to replace this description.

Nice work on your project!

Comments on "Questions, ideas, & design of analysis":
Your overall approach is to train a gradient boosting model on a training set, then use a carefully constructed holdout set to estimate accuracy on the test data. The holdout set is constructed by matching data points from the test set, essentially to choose a "control group" point for each data point in the test set. The estimated accuracy is, as expected, lower (and likely more reliable) after this matching process as compared to using the "raw" holdout set without matching.
(As an aside, you chose gradient boosting because logistic seemed to have low accuracy, <80% , in your initial experiment -- however in my code I think logistic has >95% so I'm not sure exactly how this happened).

Overall, this is a solid approach and clearly the matching shows a lot of improvement in the covariate distribution. The synthetic data experiment for verifying the improvement in uncertainty quantification has some issues (see more comments below), and it would be good to explore how the matching procedure affects the results / different options for matching / what is the best way to measure distance (more comments on "matchit" below).

Comments on "Statistical methodology":
The main process for ensuring validity is the matching approach for creating a holdout set that is more similar to the test set. This is a very solid approach to the problem of covariate shift in this data set, and the visualizations and numerical results showing the improvement in the similarity are very compelling. One issue is that the matching is carried out with the "matchit" function and it's not fully discussed what exactly is being done by this function -- for example, since the different variables are on very different scales, what is the meaning of "nearest" / of distance, in this context? In addition there may be some outliers that should be discarded (perhaps this is the reason that even after matching, some variables like koi_period are still very different).

A more important issue is that the simulation carried out with the synthetic dataset (which is intended for checking if the matched-holdout procedure is indeed accomplishing the desired result) is carried out with multiple different approaches, only some of which are targeted towards this intended goal. Specifically, the way we would want to do this is:
* Create a synthetic dataset where the training set and the test set exhibit some covariate shift . This is what your code does successfully.
* Then, run *the same method(s)* you run on the real data, on this synthetic data, to see how it performs. This appears to be what you do in the "Matching Evaluations" subsection described on page 27, but the "Weighting Evaluation" (i.e., weighted conformal) described before, and the "One side split conformal" described after, appear to be separate from any methods you ran on the real data.

Comments on "Report & code":
The report covers a good level of detail and the code is well integrated into the report. The visualizations are excellent. One issue is that of clarity in terms of presenting the high level ideas. It's clear from the report that your group's initial way of thinking about the problem was by way of conformal prediction, and then you moved away from this due to the binary label data; this is the path you followed towards designing your final procedure, but it's not the most clear

way to present the work because you are centering your discussion around a methodology (i.e., conformal) that doesn't arise in your project [except in the simulation at the end -- see comments above]. For example, on page 1 it would be more clear to give an overview of what route you took rather than describing why you didn't use conformal. Similarly on page 21 (the "Weighted Accuracy" section) you can simply say that the fraction correct in the holdout set estimates the fraction correct in the test set (where prediction is 1 if fitted probability is >0.5, or 0 otherwise), without discussing margin of error / conformal / etc. One other issue as mentioned earlier is that using a preexisting package ("matchit") should be accompanied by a more clear explanation of exactly what process the function is performing.

Project 2 Data Option

2023-03-08

In this project, we try to build a classifier to predict whether the unlabeled test points that are currently identified as "CANDIDIATE" are likely to be a real exoplanet based on other observed characteristics. Our main focus is estimating the uncertainty and accuracy of the classifier.

Initially, we thought of applying conformal inference methods to quantify uncertainty of our classifier. Instead of using residuals as the conformal score, we tried to find a different score function that is appropriate for categorical variables. We define the score as the negative of the logistic regression outcome. This is because the higher the score is, for example the residual, the worse the prediction is, while a larger logit model output means a better predictive power. Therefore choosing negative of the logistic regression outcome would align with the meaning of the scores that we want. However, we soon realized that this was not the best approach because for a binary classification problem, it would not make much sense to predict a set of labels. Typically utilizing conformal inference for classification problems means that instead of a prediction interval, we would get a set of likely labels. For this problem, there are only two possible labels, confirmed or false positive. If we try to get $1-\alpha$ coverage, our interval we may end up with both potential labels which would not tell us much about the accuracy of the classifier.

Another issue with conformal inference methods is that even if they could guarantee a $1-\alpha$ coverage, they require exchangeability of train and test points. Even if the train and test datasets have the same distribution of Y conditional on X, the distributions of X may differ which means the exchangeability assumption would not hold.

Exploration

We did some initial data cleaning, removing rows with NA values and keeping only the rows that has complete data entries.

We then try two classifiers, logistic regression and gradient boosting. We calculate the accuracy of the two classifiers by using 10 fold cross validation and find that logistic regression has 76% accuracy and gradient boosting has 90% accuracy. We also check for feature importance and see that the variables koi_prad, koi_period, koi_depth and koi_impact are the most important covariates in predicting whether the object is an exoplanet or not. However, it's worth mentioning that these accuracy numbers are actually not a good estimate for the accuracy of the classifier on the unlabeled test set because our train and test set does not have exchangeability and the covariates' distributions differ. Later we will explore the covariate shift in greater depth by checking their distributions.

```
koi_df = read.csv("KOI.csv")
analysis_df = koi_df[c("ra","dec","koi_period","koi_time0bk","koi_impact","koi_duration","koi_depth","k
analysis_df$koi_disposition = as.factor(analysis_df$koi_disposition)
analysis_df$koi_disposition <- factor(analysis_df$koi_disposition, labels = c(1,0))
analysis_df = analysis_df[complete.cases(analysis_df), ]
```

```
set.seed(123)
trainIndex <- createDataPartition(analysis_df$koi_disposition, p = .8,
                                  list = FALSE,
```

```
                                    times = 1)

training <- analysis_df[ trainIndex,]
testing  <- analysis_df[-trainIndex,]

fitControl <- trainControl(## 10-fold CV
                           method = "repeatedcv",
                           number = 10,
                           ## repeated ten times
                           repeats = 1)
```

```
# Fitting gradient boosting takes around 5 minutes
gbmFit1 <- train(koi_disposition ~ ., data = training,
                 method = "gbm",
                 trControl = fitControl,
                 verbose = FALSE)
```

```
koi_pred <- predict(gbmFit1, testing)
#confusionMatrix(data = koi_pred, testing$koi_disposition)
gbmImp <- varImp(gbmFit1, scale=FALSE)
#gbmImp
```

```
suppressWarnings(logisticFit <- train(koi_disposition ~ ., data = training,
                 method="glm",
                 family="binomial",
                 trControl = fitControl))
```

```
koi_pred <- predict(logisticFit, testing)
#confusionMatrix(data = koi_pred, testing$koi_disposition)
```

Checking for covariate shift

As discussed earlier, to use the conformal prediction methods that we learned in class, we need to check for the assumption that exchangeability holds. Specifically, the X_{n+1}, Y_{n+1} from the Unlabeled data set should follow the same distribution as $(X_1, Y_1)...(X_n, Y_n), i = 1, ..n$ from the Labeled data set. If the assumption does not hold, we need to find new ways of determining the accuracy of the classifier on the test set such as assigning weights on the data points based on some criteria.

We attempt to see whether there are significant differences between the distributions of the covariates in the labeled (train) and unlabeled (test) sets. The first thing we try is to fit a classification model across the entire dataset to predict whether a data point is labeled or unlabeled. If the classifier is able to distinguish between labeled and unlabeled points, then it suggests that the two data sets have different distributions and therefore are not exchangeable.

From running gradient boosting with 10 fold cross validation, we see that the model has 77% accuracy. Since this is much higher than 50%, and the classifier is able to predict quite well whether a data point is labeled or unlabeled, it suggests there is quite a significant difference between the two distributions. Next, we try to find out whether there are specific covariates that have different distributions between the labeled and unlabeled data.

```r
# This chunk is directly taken from file "ConformalWeights" file
set.seed(123)
trainIndex <- createDataPartition(analysis_df$koi_disposition, p = .8,
                                  list = FALSE,
                                  times = 1)

training <- analysis_df[ trainIndex,]
testing  <- analysis_df[-trainIndex,]

fitControl <- trainControl(## 10-fold CV
                           method = "repeatedcv",
                           number = 10,
                           classProbs = TRUE,
                           summaryFunction = twoClassSummary,
                           ## repeated ten times
                           repeats = 1)

gbmFit1 <- train(koi_disposition ~ ., data = training,
                 metric = "ROC",
                 method = "gbm",
                 trControl = fitControl,
                 verbose = FALSE)

koi_pred <- predict(gbmFit1, testing)
#confusionMatrix(data = koi_pred, testing$koi_disposition)
```

Determining which covariates shifted

There are multiple ways to check for which covariate have a different distribution in the train and test set. Here we include 2 methods:

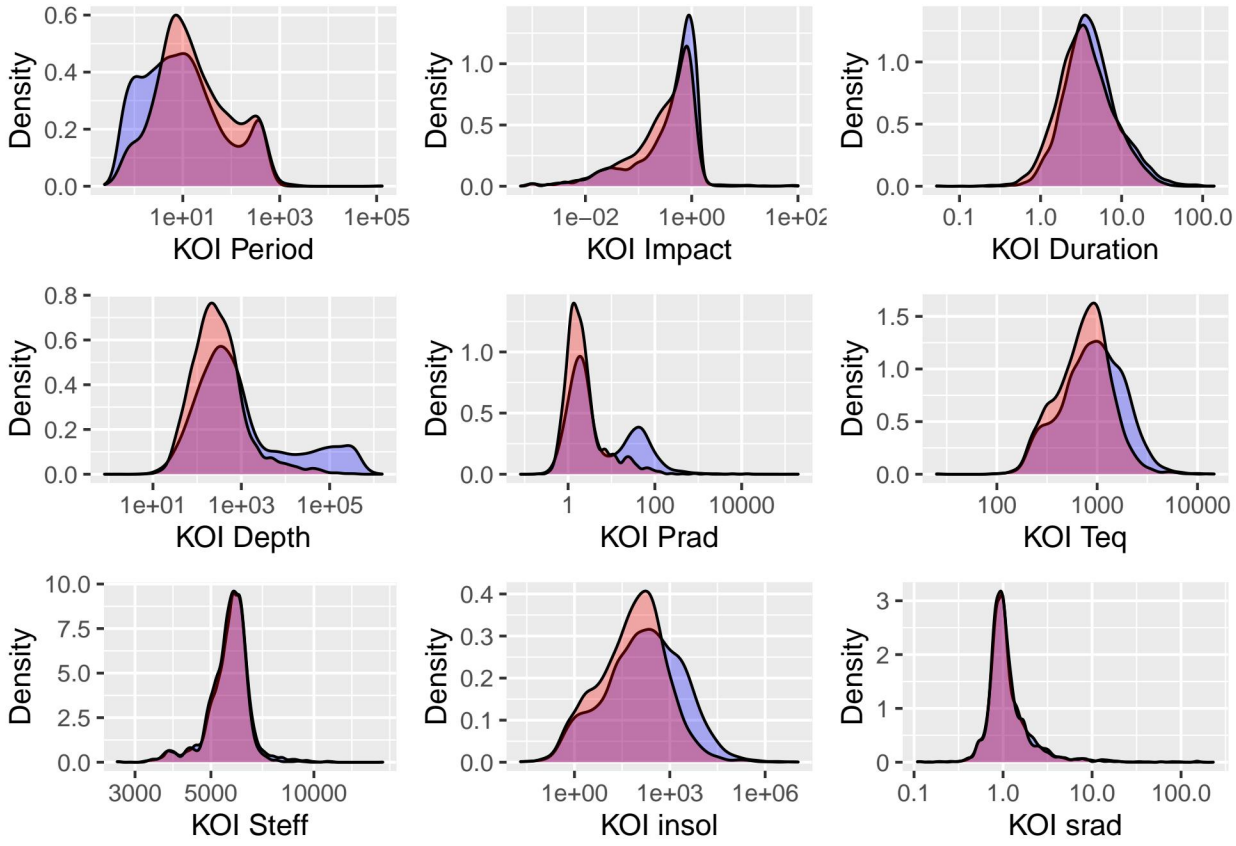
1. Visualization including density plots of all covariates that we're interested in, box plot, and scatter plots of pairwise relationship of the four covariates that we consider to be the most important in predicting koi_disposition.
2. Test statistics. We compared the characteristics such as mean, variance, ecdf etc. of the four covariates to check that there is indeed difference in the distribution.

Visualization

Here since the range of the independent values are too large with some values as small as around 0, and others as large as around 1000, to make the plot clearer, we apply the logarithmic method on the covariate. For the 9 covariates that we plotted, except for koi_steff and koi_srad, representing the photospheric temperature and the photospheric radius of the star respectively, seem to show a roughly same distribution, all other covariates are different in their distribution to some extent. Later on we tried selecting four most important covarites (koi_period, koi_impact, koi_depth, koi_prad) with respect to their power in explaining the koi_disposition and tried to match their distributions with the Unlabeled set through methods such as nearest neighbor. And we can see that these four covariates indeed have very different distributions compared with Unlabeled dataset and it's meaningful to do the matching.

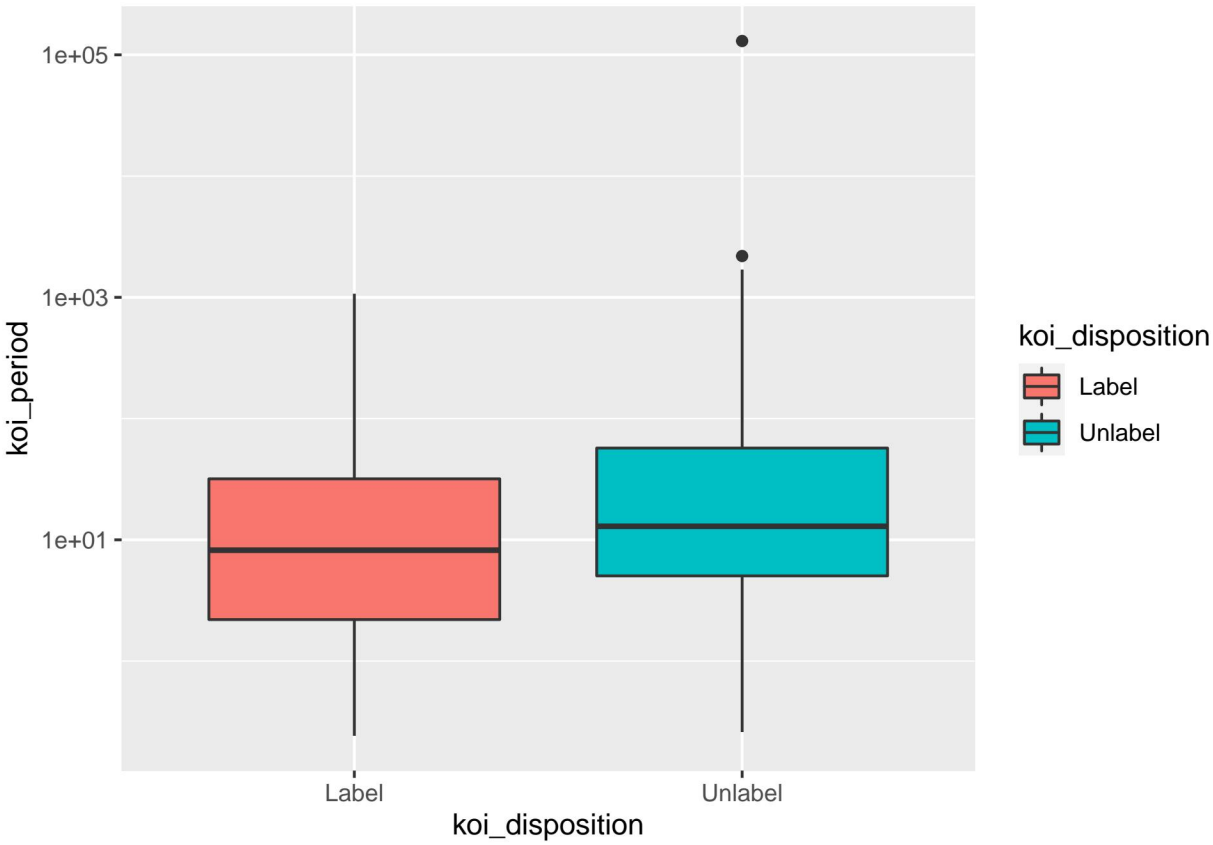
```
p_period <- ggplot() +
  geom_density(data = analysis_df, aes(x = koi_period), fill = "blue", alpha = 0.3) +
  geom_density(data = test_df, aes(x = koi_period), fill = "red", alpha = 0.3) +
  xlab("KOI Period") +
  ylab("Density") +
  scale_x_log10()

p_impact <- ggplot() +
  geom_density(data = analysis_df, aes(x = koi_impact), fill = "blue", alpha = 0.3) +
  geom_density(data = test_df, aes(x = koi_impact), fill = "red", alpha = 0.3) +
  xlab("KOI Impact") +
  ylab("Density") +
  scale_x_log10()

p_duration <- ggplot() +
  geom_density(data = analysis_df, aes(x = koi_duration), fill = "blue", alpha = 0.3) +
  geom_density(data = test_df, aes(x = koi_duration), fill = "red", alpha = 0.3) +
  xlab("KOI Duration") +
  ylab("Density") +
  scale_x_log10()

p_depth <- ggplot() +
  geom_density(data = analysis_df, aes(x = koi_depth), fill = "blue", alpha = 0.3) +
  geom_density(data = test_df, aes(x = koi_depth), fill = "red", alpha = 0.3) +
  xlab("KOI Depth") +
  ylab("Density") +
  scale_x_log10()

p_prad <- ggplot() +
  geom_density(data = analysis_df, aes(x = koi_prad), fill = "blue", alpha = 0.3) +
  geom_density(data = test_df, aes(x = koi_prad), fill = "red", alpha = 0.3) +
  xlab("KOI Prad") +
  ylab("Density") +
  scale_x_log10()

p_teq <- ggplot() +
  geom_density(data = analysis_df, aes(x = koi_teq), fill = "blue", alpha = 0.3) +
  geom_density(data = test_df, aes(x = koi_teq), fill = "red", alpha = 0.3) +
  xlab("KOI Teq") +
  ylab("Density") +
  scale_x_log10()


p_steff <- ggplot() +
  geom_density(data = analysis_df, aes(x = koi_steff), fill = "blue", alpha = 0.3) +
  geom_density(data = test_df, aes(x = koi_steff), fill = "red", alpha = 0.3) +
  xlab("KOI Steff") +
  ylab("Density") +
  scale_x_log10()

p_insol <- ggplot() +
  geom_density(data = analysis_df, aes(x = koi_insol), fill = "blue", alpha = 0.3) +
  geom_density(data = test_df, aes(x = koi_insol), fill = "red", alpha = 0.3) +
```

```
  xlab("KOI insol") +
  ylab("Density") +
  scale_x_log10()

p_srad <- ggplot() +
  geom_density(data = analysis_df, aes(x = koi_srad), fill = "blue", alpha = 0.3) +
  geom_density(data = test_df, aes(x = koi_srad), fill = "red", alpha = 0.3) +
  xlab("KOI srad") +
  ylab("Density") +
  scale_x_log10()


grid.arrange(p_period, p_impact, p_duration, p_depth, p_prad,
             p_teq, p_steff, p_insol, p_srad, ncol = 3, nrow=3)
```



The box plot below also verify that there's difference in distribution of koi_impact between the Labeled and Unlabeled data shown in the mean and variance. Since the information given by the density plot and box plot are pretty similar, here we don't include the rest of the box plot.

```
df = rbind(analysis_df, test_df)
df$koi_disposition <- ifelse(df$koi_disposition == 0 | df$koi_disposition == 1, "Label", "Unlabel")

ggplot(df, aes(x = koi_disposition, y = koi_period)) +
  geom_boxplot(aes(fill = koi_disposition)) +
  scale_y_log10()
```

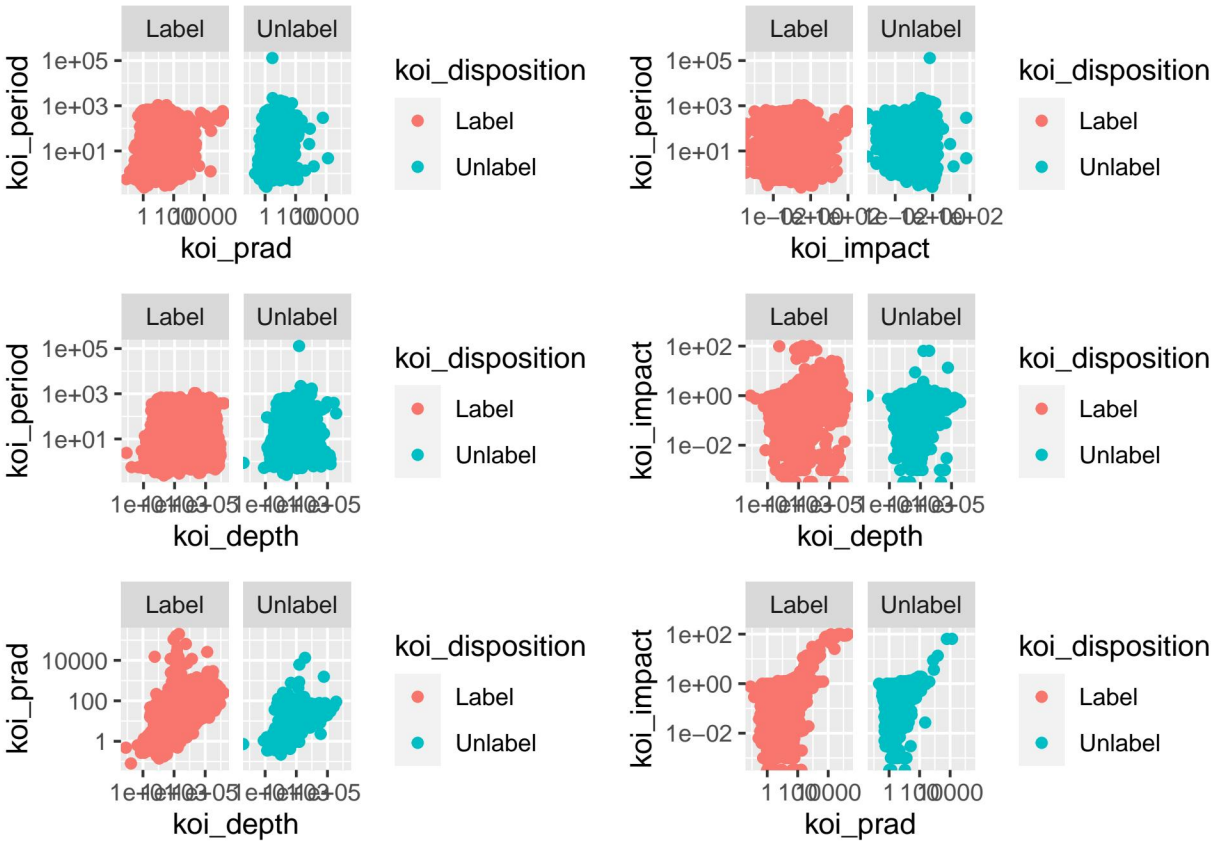


It might also be helpful to check whether the relationship differs between two covariates for the Labeled and Unlabeled data. If the relationship differs greatly, it means that the generalizability of the model is not good and the accuracy of the prediction by the model will be low. Here we looked at a few examples of the relationship between the four important covariates.

From the outcome we can see that most plots displays similar distributions but there are minor differences. For example, for the koi_period vs koi_disposition plot, we see that there's a clear positive linear relationship on the right end of the x axis (when koi_impact is larger) of the Labeled data, but there is none of the Unlabeled data.

```
par(mfrow = c(1, 2))

p1 <- ggplot(data=df, aes(x = koi_prad, y = koi_period)) +
  geom_point(aes(color = koi_disposition)) +
  facet_wrap(~koi_disposition) +
  scale_x_log10() +
  scale_y_log10()

p2 <- ggplot(data=df, aes(x = koi_impact, y = koi_period)) +
  geom_point(aes(color = koi_disposition)) +
  facet_wrap(~koi_disposition) +
  scale_x_log10() +
  scale_y_log10()

p3 <- ggplot(data=df, aes(x = koi_depth, y = koi_period)) +
  geom_point(aes(color = koi_disposition)) +
  facet_wrap(~koi_disposition) +
  scale_x_log10() +
```

```
  scale_y_log10()

p4 <- ggplot(data=df, aes(x = koi_depth, y = koi_impact)) +
  geom_point(aes(color = koi_disposition)) +
  facet_wrap(~koi_disposition) +
  scale_x_log10() +
  scale_y_log10()

p5 <- ggplot(data=df, aes(x = koi_depth, y = koi_prad)) +
  geom_point(aes(color = koi_disposition)) +
  facet_wrap(~koi_disposition) +
  scale_x_log10() +
  scale_y_log10()

p6 <- ggplot(data=df, aes(x = koi_prad, y = koi_impact)) +
  geom_point(aes(color = koi_disposition)) +
  facet_wrap(~koi_disposition) +
  scale_x_log10() +
  scale_y_log10()

grid.arrange(p1, p2, p3, p4, p5, p6, ncol = 2, nrow=3)
```



Then we check the balance between the labeled and unlabeled data. For the four variables that were the most used in classification during exploratory analysis we find large differences between the datasets, confirming the fact that their distributions are different as we observed in the visualization. The variables we look at are koi_prod, koi_period, koi_impact, and koi_depth. Each variable has imbalance as assessed by difference

in means, variance ratio, standardized mean differrence, and eCDF. We summarize the imbalances with a table below.

```
# Create formula for matching based on train variable and covariates

formula <- train ~ koi_prad + koi_period + koi_impact + koi_depth

# Checking balance

match_object_before <- matchit(formula, data = df, method = NULL, distance="glm")
summary(match_object_before)
```

```
##
## Call:
## matchit(formula = formula, data = df, method = NULL, distance = "glm")
##
## Summary of Balance for All Data:
##            Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean
## distance          0.2691        0.2276          1.1109     0.1674    0.1500
## koi_prad         15.9315      129.9739         -0.3598     0.0081    0.1464
## koi_period      131.2826       56.5643          0.0268   538.8348    0.0881
## koi_impact        0.5371        0.7968         -0.1305     0.2942    0.0842
## koi_depth      1864.2389    30620.1140         -2.3001     0.0181    0.1474
##            eCDF Max
## distance     0.2328
## koi_prad     0.2361
## koi_period   0.1646
## koi_impact   0.1349
## koi_depth    0.2210
##
## Sample Sizes:
##           Control Treated
## All          7016    2185
## Matched      7016    2185
## Unmatched       0       0
## Discarded       0       0
```

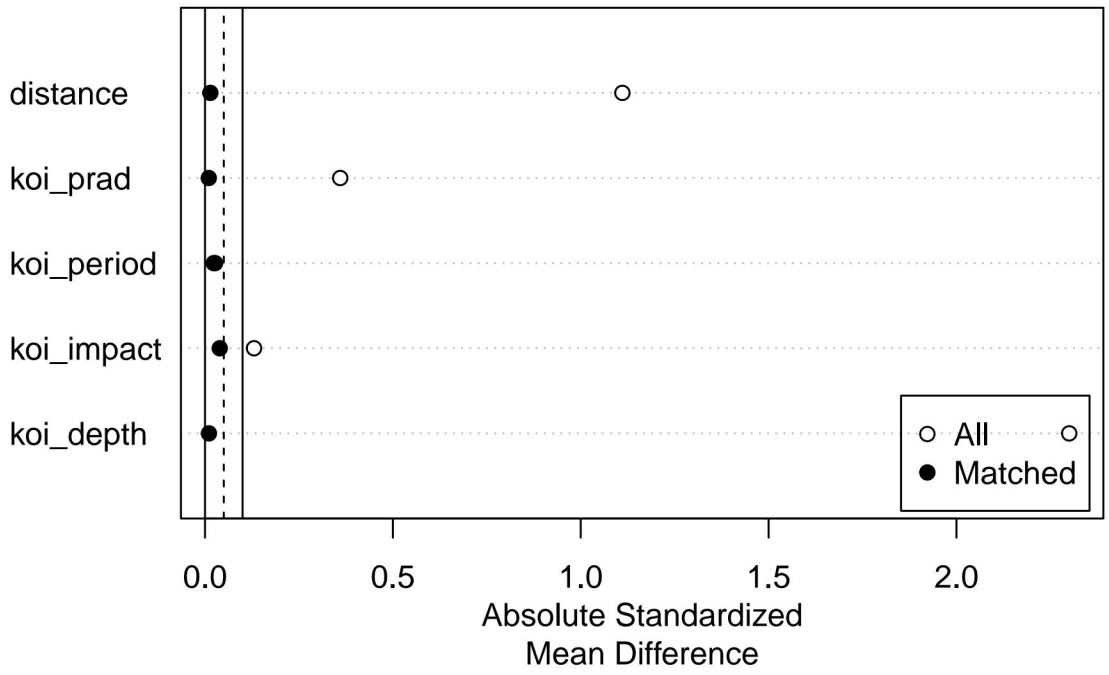
K Nearest Neighbor matching to create a holdout set similar to the test set

We try several different matching techniques. The first is greedy nearest neighbor matching, which involves calculating the distance between every treated unit and control unit. Then, one at a time, each treated unit is paired with a control unit as its match. The matching process is "greedy" because it doesn't optimize. Each match is made without considering any potential matches that may come later in the process. In the table below and in the graph densities we see a much better balance between the covariates, although they are still not ideal.

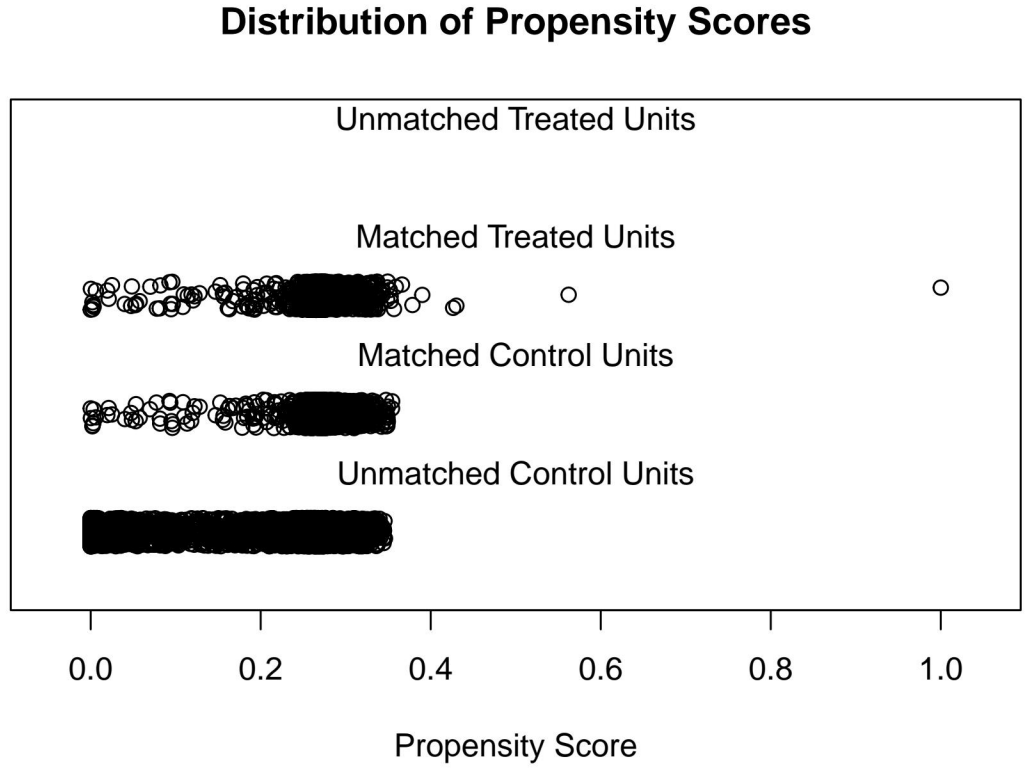
```
# Create MatchIt object using nearest neighbor matching
match_object <- matchit(formula, data = df, method = "nearest")

# Summarize the MatchIt object
summary(match_object)
```

```
##
## Call:
## matchit(formula = formula, data = df, method = "nearest")
##
## Summary of Balance for All Data:
##            Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean
## distance          0.2691        0.2276          1.1109     0.1674    0.1500
## koi_prad         15.9315      129.9739         -0.3598     0.0081    0.1464
## koi_period      131.2826       56.5643          0.0268   538.8348    0.0881
## koi_impact        0.5371        0.7968         -0.1305     0.2942    0.0842
## koi_depth      1864.2389    30620.1140         -2.3001     0.0181    0.1474
##            eCDF Max
## distance     0.2328
## koi_prad     0.2361
## koi_period   0.1646
## koi_impact   0.1349
## koi_depth    0.2210
##
## Summary of Balance for Matched Data:
##            Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean
## distance          0.2691        0.2686          0.0142     1.2812    0.0002
## koi_prad         15.9315       12.7424          0.0101     5.7818    0.0212
## koi_period      131.2826       66.1910          0.0234   463.3374    0.0640
## koi_impact        0.5371        0.4592          0.0392    26.3959    0.0229
## koi_depth      1864.2389     1996.1459         -0.0106     0.8677    0.0213
##            eCDF Max Std. Pair Dist.
## distance     0.0046          0.0145
## koi_prad     0.0792          0.0705
## koi_period   0.1400          0.0332
## koi_impact   0.0595          0.1390
## koi_depth    0.0645          0.0587
##
## Sample Sizes:
##           Control Treated
## All          7016    2185
## Matched      2185    2185
## Unmatched    4831       0
## Discarded       0       0
```

```
plot(summary(match_object))
```

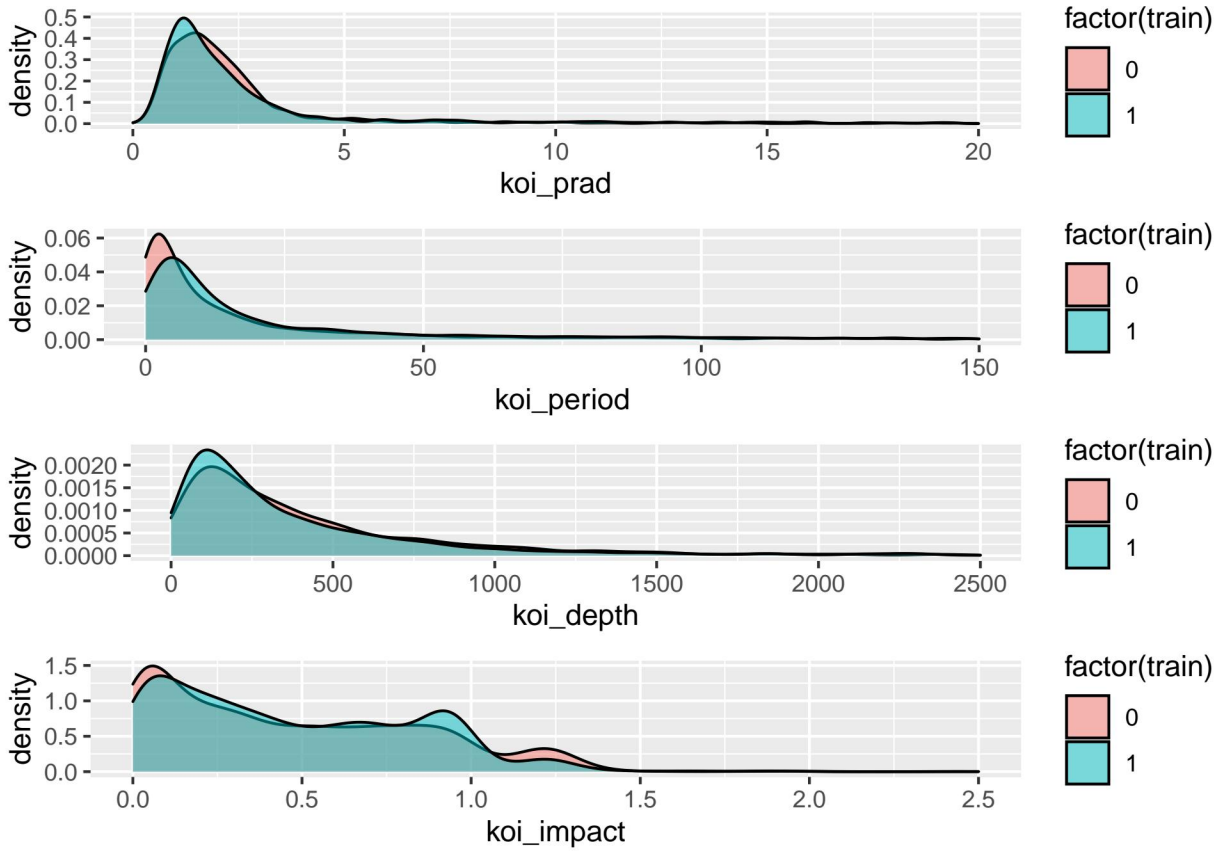



```
# Plot
plot(match_object, type = "jitter", interactive = FALSE)
```




```
# Extract matched data
matched_data_nearest <- match.data(match_object)
```

```
df <- matched_data_nearest

plot1_after <- ggplot(df, aes(x = koi_prad, fill= factor(train))) +  geom_density(position="identity", a

plot2_after <- ggplot(df, aes(x = koi_period, fill= factor(train))) +  geom_density(position="identity"

plot3_after <- ggplot(df, aes(x = koi_depth, fill= factor(train))) +  geom_density(position="identity",

plot4_after <- ggplot(df, aes(x = koi_impact, fill= factor(train))) +  geom_density(position="identity"

grid.arrange(plot1_after, plot2_after, plot3_after, plot4_after,ncol = 1)
```



Trying alternative approaches to matching

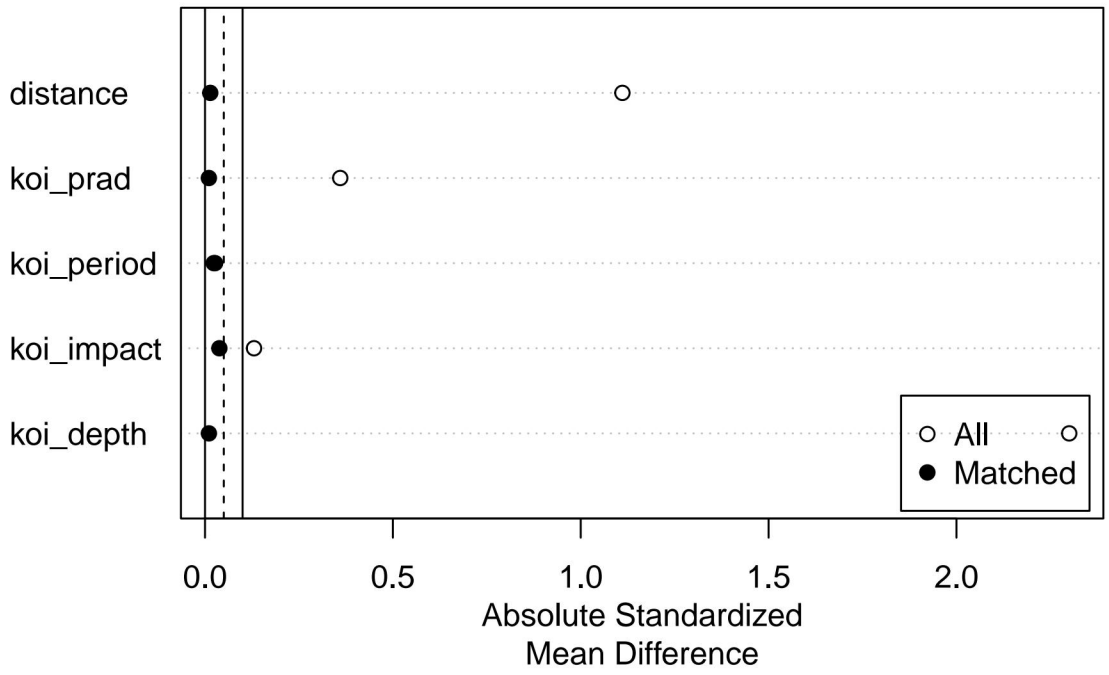
Below we try two improvements to the matching procedure. First, we try an "optimal" procedure that minimizes the sum of the absolute pairwise distances in teh matched sample. Then we perform optimal full matching, in which all units, both unlabeled and labeled, receive at least one match. The optimal procedure is a downsampling methethod and the full matching procedure is an upsampling mthod, which samples with replacement. Upsampling and downsampling are strategies to deal with imbalanced data sets where one group is much larger than the other. Upsampling means increasing the size of the smaller group by duplicating or creating new observations, while downsampling means decreasing the size of the larger group by removing or combining observations.

Optimal

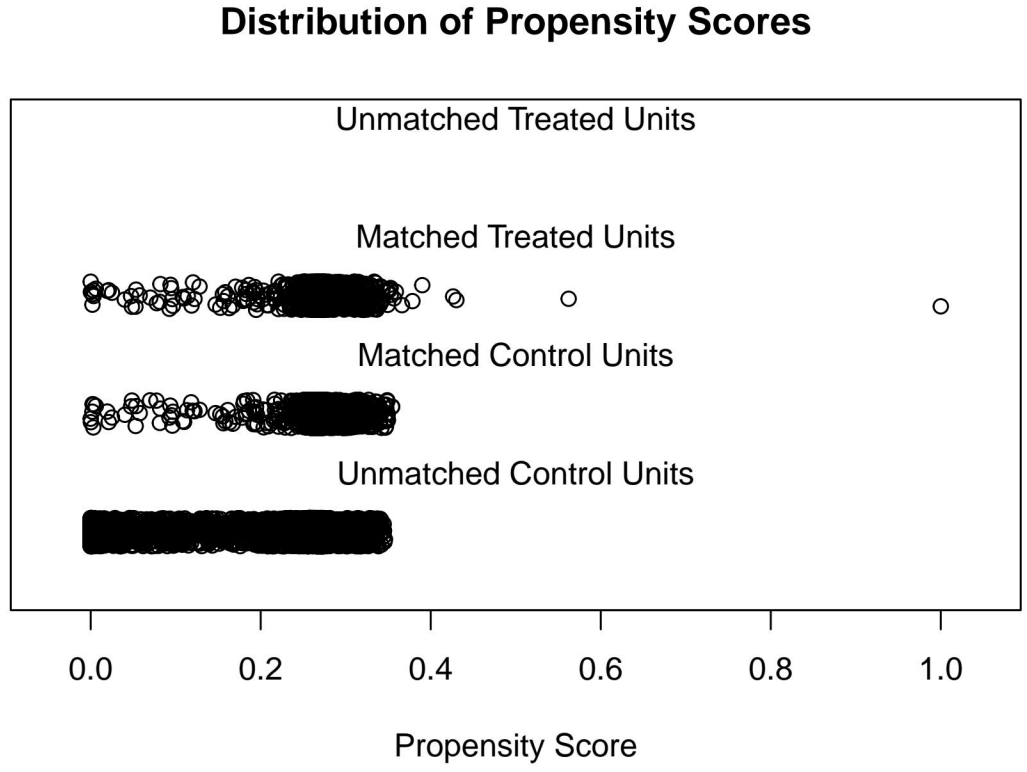
```
#This chunk takes about 5 mins to run
#formula <- train ~ koi_prad + koi_period + koi_impact + koi_depth
df <- df_t
# Create MatchIt object using nearest neighbor matching
match_object <- matchit(formula, data = df, method = "optimal")

# Summarize the MatchIt object
summary(match_object)
```

```
##
## Call:
## matchit(formula = formula, data = df, method = "optimal")
##
## Summary of Balance for All Data:
##            Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean
## distance          0.2691        0.2276          1.1109     0.1674    0.1500
## koi_prad         15.9315      129.9739         -0.3598     0.0081    0.1464
## koi_period      131.2826       56.5643          0.0268   538.8348    0.0881
## koi_impact        0.5371        0.7968         -0.1305     0.2942    0.0842
## koi_depth      1864.2389    30620.1140         -2.3001     0.0181    0.1474
##            eCDF Max
## distance     0.2328
## koi_prad     0.2361
## koi_period   0.1646
## koi_impact   0.1349
## koi_depth    0.2210
##
## Summary of Balance for Matched Data:
##            Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean
## distance          0.2691        0.2686          0.0139     1.2809    0.0002
## koi_prad         15.9315       12.6575          0.0103     5.7841    0.0205
## koi_period      131.2826       66.4518          0.0233   460.4299    0.0635
## koi_impact        0.5371        0.4611          0.0382    26.4394    0.0212
## koi_depth      1864.2389     1994.0122         -0.0104     0.8676    0.0203
##            eCDF Max Std. Pair Dist.
## distance     0.0046          0.0145
## koi_prad     0.0792          0.0702
## koi_period   0.1400          0.0333
## koi_impact   0.0572          0.1377
## koi_depth    0.0600          0.0586
##
## Sample Sizes:
##           Control Treated
## All          7016    2185
## Matched      2185    2185
## Unmatched    4831       0
## Discarded       0       0
```

```
plot(summary(match_object))
```



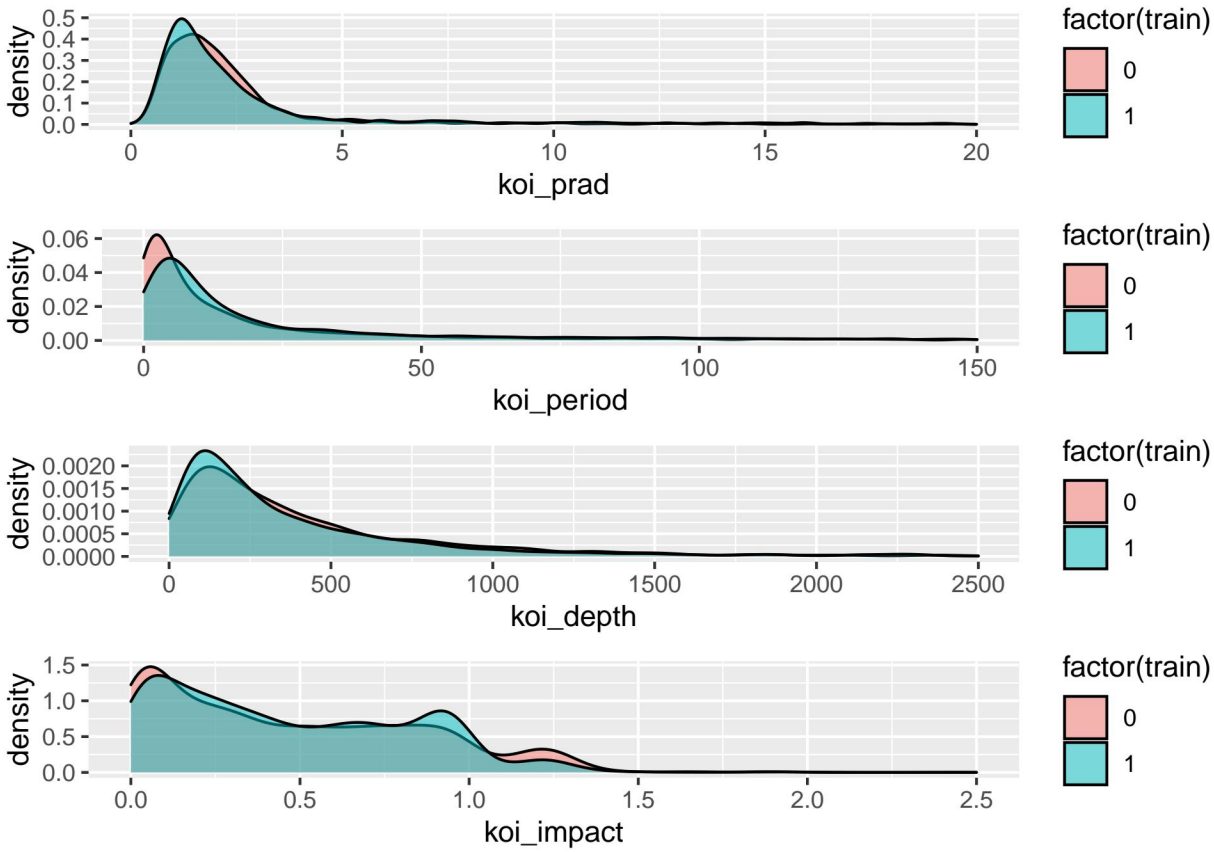

```
# Plot
plot(match_object, type = "jitter", interactive = FALSE)
```




```
# Extract matched data
matched_data_optimal <- match.data(match_object)
```

```
df <- matched_data_optimal

plot1_after <- ggplot(df, aes(x = koi_prad, fill= factor(train))) +  geom_density(position="identity", a

plot2_after <- ggplot(df, aes(x = koi_period, fill= factor(train))) +  geom_density(position="identity"

plot3_after <- ggplot(df, aes(x = koi_depth, fill= factor(train))) +  geom_density(position="identity",

plot4_after <- ggplot(df, aes(x = koi_impact, fill= factor(train))) +  geom_density(position="identity"

grid.arrange(plot1_after, plot2_after, plot3_after, plot4_after,ncol = 1)
```

```
## Warning: Removed 364 rows containing non-finite values (stat_density).

## Warning: Removed 645 rows containing non-finite values (stat_density).

## Warning: Removed 271 rows containing non-finite values (stat_density).

## Warning: Removed 5 rows containing non-finite values (stat_density).
```

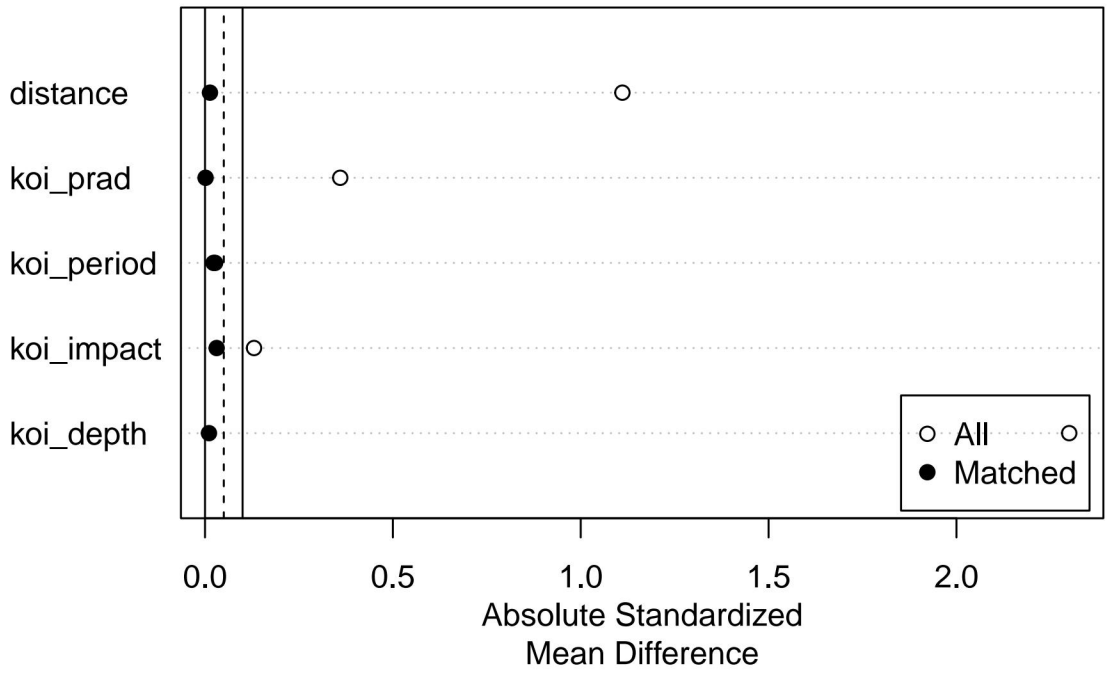


Full

```r
# This took around 5 minutes to run

df <- df_t

#formula <- train ~ koi_prad + koi_period + koi_impact + koi_depth

#plot(summary(match_object_before))

# Create MatchIt object using nearest neighbor matching
match_object <- matchit(formula, data = df, method = "full")

# Summarize the MatchIt object
summary(match_object)
```

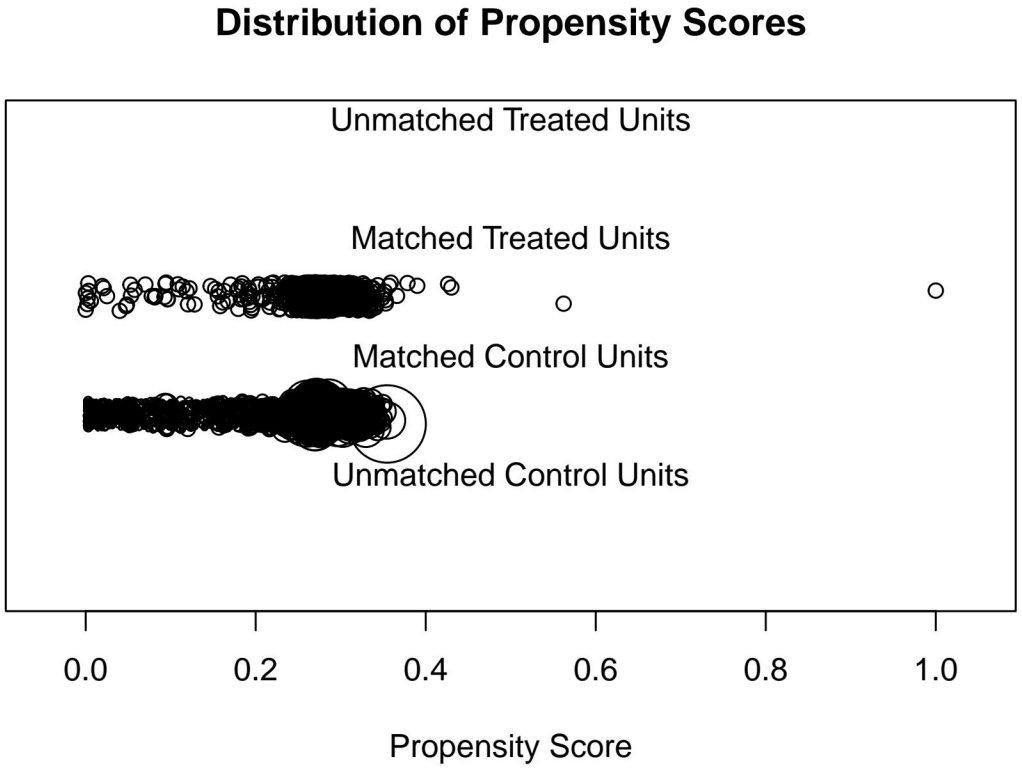
```
##
## Call:
## matchit(formula = formula, data = df, method = "full")
##
## Summary of Balance for All Data:
##            Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean
## distance          0.2691        0.2276          1.1109     0.1674    0.1500
## koi_prad         15.9315      129.9739         -0.3598     0.0081    0.1464
## koi_period      131.2826       56.5643          0.0268   538.8348    0.0881
## koi_impact        0.5371        0.7968         -0.1305     0.2942    0.0842
## koi_depth      1864.2389    30620.1140         -2.3001     0.0181    0.1474
##            eCDF Max
## distance     0.2328
## koi_prad     0.2361
## koi_period   0.1646
## koi_impact   0.1349
## koi_depth    0.2210
##
## Summary of Balance for Matched Data:
##            Means Treated Means Control Std. Mean Diff. Var. Ratio eCDF Mean
## distance          0.2691        0.2686          0.0132     1.2761    0.0002
## koi_prad         15.9315       16.4042         -0.0015     0.1426    0.0221
## koi_period      131.2826       67.8781          0.0228   447.2480    0.0619
## koi_impact        0.5371        0.4754          0.0310     4.9132    0.0207
## koi_depth      1864.2389     1995.7038         -0.0105     0.7815    0.0253
##            eCDF Max Std. Pair Dist.
## distance     0.0041          0.0123
## koi_prad     0.0768          0.6648
## koi_period   0.1398          0.0230
## koi_impact   0.0517          0.4685
## koi_depth    0.0655          0.5432
##
## Sample Sizes:
##               Control Treated
## All           7016.      2185
## Matched (ESS) 2604.25    2185
## Matched       7016.      2185
## Unmatched        0.         0
## Discarded        0.         0
```

```
plot(summary(match_object))
```



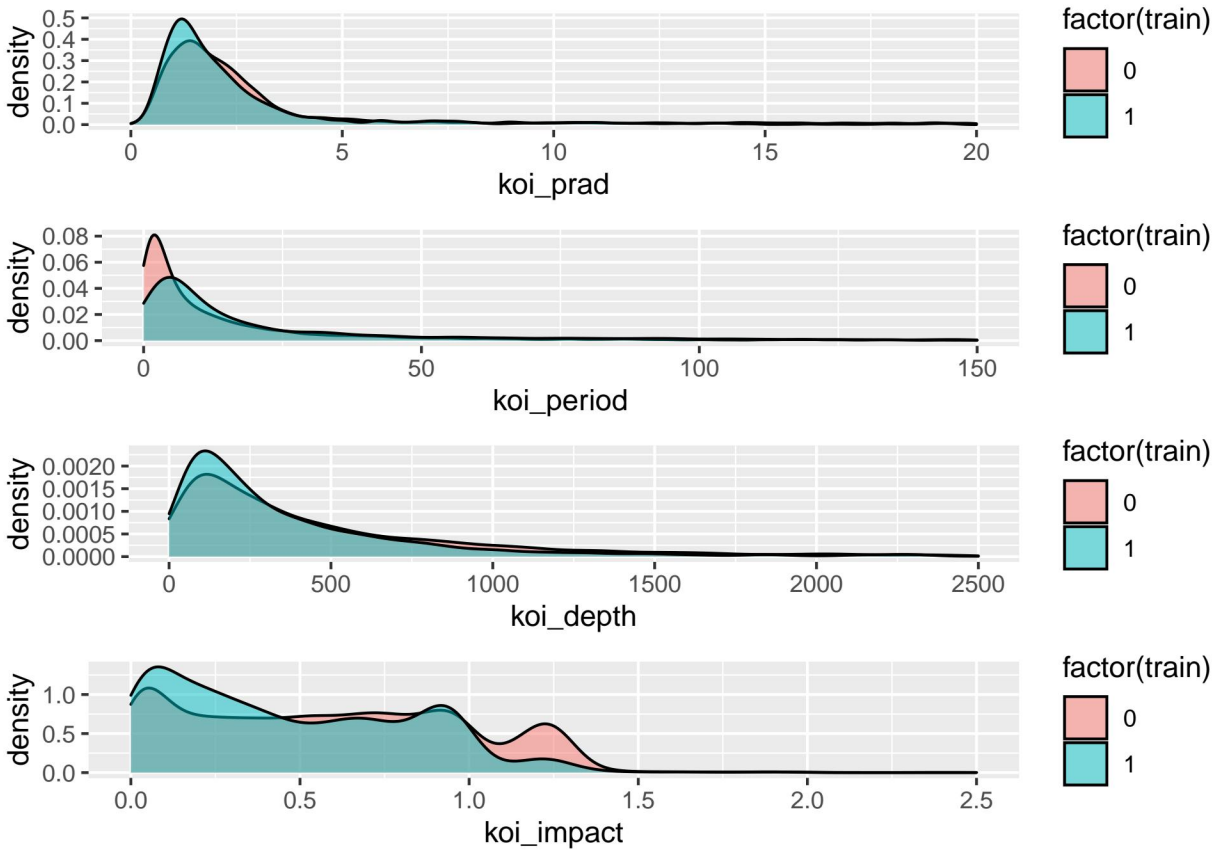

```
# Plot
plot(match_object, type = "jitter", interactive = FALSE, ncol = 1)
```

```
## Warning in plot.window(...): "ncol" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "ncol" is not a graphical parameter

## Warning in title(...): "ncol" is not a graphical parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "ncol" is not a graphical
## parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "ncol" is not a graphical
## parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "ncol" is not a graphical
## parameter

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "ncol" is not a graphical
## parameter
```

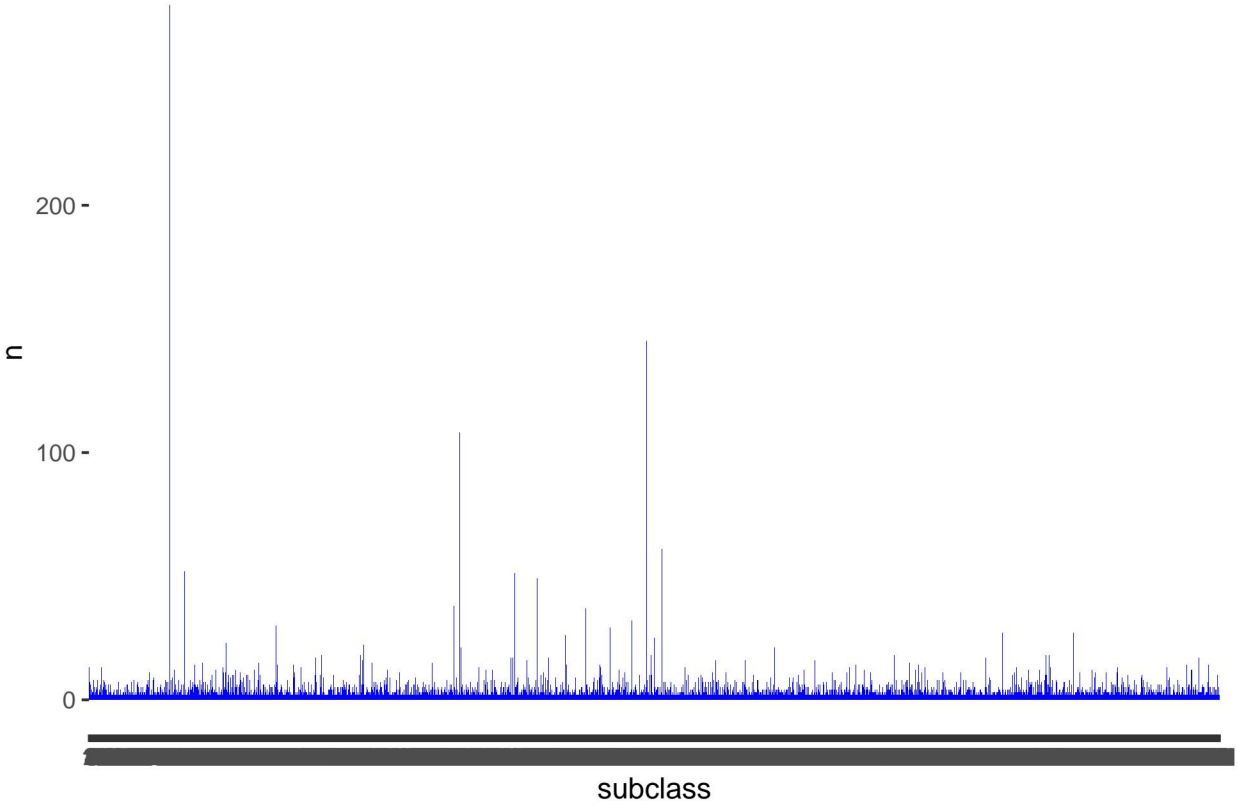



```
# Extract matched data
matched_data_full <- match.data(match_object)
```

```
df <- matched_data_full

plot1_after <- ggplot(df, aes(x = koi_prad, fill= factor(train))) +  geom_density(position="identity", a

plot2_after <- ggplot(df, aes(x = koi_period, fill= factor(train))) +  geom_density(position="identity"


plot3_after <- ggplot(df, aes(x = koi_depth, fill= factor(train))) +  geom_density(position="identity",


plot4_after <- ggplot(df, aes(x = koi_impact, fill= factor(train))) +  geom_density(position="identity"

grid.arrange(plot1_after, plot2_after, plot3_after, plot4_after,ncol = 1)
```



With full matching, we see that often a labeled observation is matched to multiple unlabeled observations, i.e., upsampling. For further analysis we prefer the downsampled sample because it preserves the general trend of the signal (although it increases uncertainty by removing many labeled observations).



Using the matched dataset

After using nearest neighbors to create a matched dataset, we check again for exchangeability once more by seeing whether a classifier can accurately predict whether a datapoint is labeled or unlabeled. We see that the accuracy has gone down significantly to 63% comparing to 77% prior to matching. This suggests that our matched dataset is more likely to have exchangeability with our unlabeled test data.

```
matched_data_nearest = matched_data_nearest[matched_data_nearest$koi_disposition != "CANDIDATE",]
write.csv(matched_data_nearest,"matched_data_nearest.csv")
total = bind_rows(test_set, matched_data_nearest)
total = total[,1:14]
```

```
# This chunk is directly taken from file "ConformalWeights"
set.seed(123)
new_df = total %>%
                  mutate(koi_disposition = recode(koi_disposition,
                                    "CONFIRMED" = "NC",
                                    "FALSE POSITIVE" = "NC",
                                    "CANDIDATE" = "C"))
total$koi_disposition = as.factor(new_df$koi_disposition)
analysis_df = total[complete.cases(total), ]

trainIndex <- createDataPartition(analysis_df$koi_disposition, p = .8,
                                  list = FALSE,
                                  times = 1)
```

```
training <- analysis_df[ trainIndex,]
testing  <- analysis_df[-trainIndex,]

fitControl <- trainControl(## 10-fold CV
                           method = "repeatedcv",
                           number = 10,
                           classProbs = TRUE,
                           summaryFunction = twoClassSummary,
                           ## repeated ten times
                           repeats = 1)

gbmFit1 <- train(koi_disposition ~ ., data = training,
                 metric = "ROC",
                 method = "gbm",
                 trControl = fitControl,
                 verbose = FALSE)

koi_pred <- predict(gbmFit1, testing)
#confusionMatrix(data = koi_pred, testing$koi_disposition)
```

Now, we try using the matched data as a holdout set and train a model on the remaining labeled data. When using the matched data as the holdout set, we see a reduction in the accuracy of the gradient boosting classifier. When we initially calculated the accuracy using 10 fold cross validation, we got an accuracy of 90%. When we test the classifier on the matched holdout set, we get a lower level of accuracy at 85%. This is expected because our classifier would not work as well on the test set if there is a distribution shift. Therefore, using a holdout set that is matched to the test data in terms of 4 different covariates seems to get us a more realistic measure of accuracy even if it is lower. However, this method means that we reduce the size of our training set by the points that most resemble the test set which is not ideal as we would like to train the classifier on as much of our training data as possible.

```
matched_data_nearest = matched_data_nearest[matched_data_nearest$koi_disposition != "CANDIDATE",]
validation_set = matched_data_nearest[,1:14]

validation_set$koi_disposition = as.factor(validation_set$koi_disposition)
validation_set$koi_disposition <- factor(validation_set$koi_disposition, labels = c(1,0))
#create training set (excluding validation set)
index = analysis_df$koi_period  %in% validation_set$koi_period
train_set = analysis_df[!index,]
```

```
# Fitting gradient boosting
gbmFit1 <- train(koi_disposition ~ ., data = train_set,
                 method = "gbm",
                 verbose = FALSE)
koi_pred <- predict(gbmFit1, validation_set)
#confusionMatrix(data = koi_pred, validation_set$koi_disposition)
```

Weighted Accuracy

Another approach we attempt for quantifying the uncertainty of the classifier involveS working from the other direction than a typical conformal method. Instead of determining a margin of error from a fixed accuracy level, we instead tried to find a bound for the accuracy level given a fixed margin of error. The way that this approach would work in the continuous case is that given $\hat{u}$ and a fixed margin of error Δ we would try to find the accuracy of $\hat{u} \pm \Delta$. Given a training set with points $1, 2, ..., n_0$ and a holdout set with points $n_0, n_0 + 1, ..., n$, the accuracy to determine would be:

$$\frac{\sum_{i=n_0+1}^{n} 1_{Y_i \in \hat{u}(x_i) \pm \Delta}}{n - n_0}$$

Note that these holdout points would not be the points for which we are actually trying to estimate the uncertainty for - they would just be the labeled points being held out from training.

However this approach must be modified for the setting of binary classification. Δ cannot be a real number since it is not very meaningful to state that the data point was an exoplanet $\pm\Delta$. We can only think of two scenarios in which this makes sense. The first is if the classification problem were converted into a problem of estimating the probability that the point was in either class. However this simply converts the classification problem into a continuous prediction question and then any weighted version of conformal or split conformal prediction should perform well. Moreover in an actual dataset we will always just have the final classification value - not the probability that the point would fall into that category so converting the binary classifications fully into probabilities is not useful in practice. However one approach we try later is to try to find the smallest cut off probability for assigning a point as an exoplanet as possible while still retaining $1 - \alpha$ coverage. More details about this method are given later.

First we instead attempt to find a non-numeric representation of a margin of error. The method that we settled on was to take the margin of error as whether the probability outputted from the model was closer to the correct classification than the false classification (taking the classification of "CONFIRMED" as 1 and "FALSE POSITIVE" as 0). In practice, calculating the accuracy of this fixed margin of error ends up being the same as calculating the accuracy rate on held-out data. Held-out data is used as an analogue to split conformal: the differences being that instead of calculating a margin of error given a desired accuracy level we are calculating an accuracy for our margin of error and the margin of error being used is whether the predicted probability is within 0.5 of the real value (either 0 or 1). Since the accuracies outputted by the models lie in (0,1), this ends up being the same as the misclassification rate.

Synthetic Data

Another difficulty we encounter is that there are no ground-truth value for the unlabeled data. Thus there is no way to truly test the validity of our methods using the given unlabeled data. In order to get a sense for the validity of our methods, we construct a simulated version of the dataset as is discussed below. This process relies on a key assumption: that the distribution of the response given the covariates is constant between the labeled and unlabeled data. This assumption is necessary as otherwise there would be nothing we could do to determine the accuracy of our model on unlabeled data since the new distribution of $Y_i|X_{i,1},...X_{i,n}$ could be arbitrarily different than the original distribution. As a clear example of this say, say (unknownst to us) the original true distribution of $Y_i|X_{i,1},...X_{i,n}$ is $f_1(X_{i,1},...X_{i,n})$ and on the test distribution the true function predicts exactly the opposite label of whatever the original $f_1(X_{i,1},...X_{i,n})$ predicts. Then if we correctly fit the true model on the training distribution, we would have an accuracy of 0 on the test distribution. In general it is possible to construct a changed function with arbitrarily bad accuracy given our margin of error and so we assume that the true distribution of $f(X_{i,1},...X_{i,n})$ is constant for the simulation.

Unfortunately the true $f(X_{i,1},...X_{i,n})$ for the dataset is unknown to us, but it can be approximated by fitting a function to a subset of the labeled data and taking that function as our new true $f(X_{i,1},...X_{i,n})$. The problem then becomes that we cannot just keep the labels on the labeled data and use the function we fit to predict the labels on the unlabeled data because then $f(X_{i,1},...X_{i,n})$ would be different between the labeled unlabeled data. Instead what we do is use the classifier we fit to create simulated labels for both the remainder of the labeled data not used for training the model and the unlabeled data. We then use all the data with synthetic labels as a new dataset to estimate the effectiveness of our methods on.

```
# Initial data processing
koi_df = read.csv("KOI.csv")
analysis_df = koi_df[c("ra","dec","koi_period","koi_time0bk","koi_impact",
                       "koi_duration","koi_depth","koi_prad","koi_teq","koi_insol",
                       "koi_steff","koi_slogg","koi_srad",
                       "koi_disposition")][koi_df$koi_disposition != "CANDIDATE",]
analysis_df = analysis_df[complete.cases(analysis_df), ]
analysis_df$koi_disposition = as.factor(analysis_df$koi_disposition)
analysis_df$koi_disposition <- factor(analysis_df$koi_disposition, labels = c(1,0))

# Train complex classifier on half_analysis
half_analysis = analysis_df[1:3000,]
# Predict the labels on the other half of the labeled data h2_analysis
h2_analysis = analysis_df[-c(1:3000),]
# Also subset the unlabeled data
unlabeled_df = koi_df[c("ra","dec","koi_period","koi_time0bk","koi_impact",
                        "koi_duration","koi_depth","koi_prad","koi_teq",
                        "koi_insol","koi_steff","koi_slogg","koi_srad",
                        "koi_disposition")][koi_df$koi_disposition == "CANDIDATE",]
unlabeled_df = unlabeled_df[complete.cases(unlabeled_df), ]
```

Since we are creating the data labels with a much simpler method than the true data-generating method, we take several steps to ensure that the models being evaluated on the synthetic data are not able to exactly fit the true $f(X_{i,1},...X_{i,n})$. The first step we take to ensure this is to use a penalized logistic regression model which includes all interaction terms between covariates to create the synthetic labels and evaluate a logistic regression model without interaction terms on this dataset. We also tune the regularization parameters of our data generating process to ensure that several interaction terms between covariates display relatively large coefficients in relation to the other coefficients present in the model (on the order of 10^{-2}). These steps ensure that the models being evaluated on this data were are able to perfectly fit the data-generating function.

```
set.seed(1)
# + I(log(koi_period)) + I(log(koi_impact)
```

```r
x <- model.matrix(koi_disposition ~ .^2, data = half_analysis)[,-1] # remove intercept column
y <- half_analysis$koi_disposition
set.seed(2)
complex_model <- glmnet(x, y, family = "binomial", alpha = 0.1, lambda = 0.0001)
```

```r
# Display the coefficients
#coef(complex_model, s = 0.1)
```

```r
x_pred <- model.matrix(koi_disposition ~ .^2, data = h2_analysis)[,-1]
h2_analysis$koi_disposition = factor(as.vector(predict(complex_model,x_pred)) > 0.5,
                                     labels = c(0,1))
x_pred <- model.matrix(koi_disposition ~ .^2, data = unlabeled_df)[,-1]
unlabeled_df$koi_disposition = factor(as.vector(predict(complex_model,x_pred)) > 0.5,
                                      labels=c(0,1))
```

```r
sim_train = h2_analysis[1:1500,]
sim_test = h2_analysis[-c(1:1500),]
x <- model.matrix(koi_disposition ~ ., data = sim_train)[,-1] # remove intercept column
y <- sim_train$koi_disposition
set.seed(3)
simple_model <- glmnet(x, y, family = "binomial", alpha = 0.05, lambda = 0.1)
```

```r
# make predictions on sim_test data
x_test <- model.matrix(koi_disposition ~ ., data = sim_test)[,-1]
y_test <- sim_test$koi_disposition
pred_prob <- predict(simple_model, x_test, type = "response")

# This part is for the split-conformal type method discussed later
p_df = data.frame(class = as.double(y_test)-1, p = pred_prob)

p_df = p_df[order(-p_df$s0),]

max_i = 0

for (i in c(1:length(p_df$s0))) {
  subset <- p_df[c(1:i),]
  if (sum(subset$class)/length(subset$class) < 0.9) {
    max_i = i
    break
  }
}

#print(p_df[max_i + 1,]$s0)

# classify observations based on a threshold
threshold <- 0.5
pred_class <- ifelse(pred_prob > threshold, 1, 0)

# create a confusion matrix and calculate accuracy
conf_mat <- table(y_test, pred_class)
accuracy <- sum(diag(conf_mat)) / sum(conf_mat)
#conf_mat
#accuracy
```

```r
# make predictions on sim_test data
x_test <- model.matrix(koi_disposition ~ ., data = unlabeled_df)[,-1]
y_test <- unlabeled_df$koi_disposition
pred_prob <- predict(simple_model, x_test, type = "response")

# This part is for the split-conformal type method discussed later
p_df = data.frame(class = as.double(y_test)-1, p = pred_prob)

p_df = p_df[order(-p_df$s0),]

max_i = 0

for (i in c(1:length(p_df$s0))) {
  subset <- p_df[c(1:i),]
  if (sum(subset$class)/length(subset$class) < 0.9) {
    max_i = i
    break
  }
}

#print(p_df[max_i + 1,]$s0)

# classify observations based on a threshold
threshold <- 0.5
pred_class <- ifelse(pred_prob > threshold, 1, 0)

# create a confusion matrix and calculate accuracy
conf_mat <- table(y_test, pred_class)
accuracy <- sum(diag(conf_mat)) / sum(conf_mat)
#conf_mat
#accuracy
```

Weighting Evaluation

Without accounting for covariate shift, the logistic model without interaction terms achieves 82.83% accuracy on held-out-originally-labeled synthetic data and 67.87% accuracy on the synthetic labels for the unlabeled data. This accuracy difference is a clear sign of the change in covariate distribution leading to different uncertainty levels on the training and test data. Using the accuracy of the classifier as the uncertainty estimate for the held-out data does not seem to work well; we would say that we are 82.83% confident about each of our labels when we are truly only correct 67.87% of the time.

The first way we attempt to solve this is through substituting a weighted accuracy measure instead of the bare accuracy for the uncertainty on the test set. The weights are constructed so as to put higher weight on points similar to the test set and lower weights on point very dissimilar to the test set. This is accomplished through an adopted version of the weighting method used in Barber et al. 2019.

First a separate logistic regression is fit to predict whether the synthetic data comes from the training or test set. The weights are constructed from these probabilities by:

$$w_{i,pre} = \frac{p_i}{1 - p_i}$$

$$w_i = \frac{(n - n_0) * w_{i,pre}}{\sum_{i=n_0+1}^{n} w_{i,pre}}$$

where p_i is the probability that point i is unlabeled, $n - n_0$ is the number of datapoints in the holdout set for the classifier which we desire the uncertainty of, and w_i is the weight to multiply that point by to achieve the weighted accuracy. Thus the weighted accuracy is given by:

$$\frac{\sum_{i=n_0+1}^{n} w_i 1_{Y_i \in \hat{u}(x_i) \pm \Delta}}{n - n_0}$$

```
weight_1 <- sim_test
weight_2 <- unlabeled_df

weight_1$koi_disposition = 0
weight_2$koi_disposition = 1
weight_df = rbind(weight_1, weight_2)
weight_df$koi_disposition = factor(weight_df$koi_disposition)
```

```
x <- model.matrix(koi_disposition ~ .^2, data = weight_df)[,-1]
# remove intercept column
x_1 <- model.matrix(koi_disposition ~ .^2, data = weight_1)[,-1]
# remove intercept column
y <- weight_df$koi_disposition
# perform cross-validation with cv.glmnet
set.seed(4)
weight_model <- cv.glmnet(x, y, family = "binomial", alpha = 0.05,nfolds=5)

predicted <- predict(weight_model, x, type = "response", s = "lambda.min")
# use lambda that gives minimum mean

# classify observations based on a threshold
threshold <- 0.5
pred_class <- ifelse(predicted > threshold, 1, 0)
# create a confusion matrix and calculate accuracy
conf_mat <- table(y, pred_class)
#sum(diag(conf_mat)) / sum(conf_mat)
#conf_mat
```

```
#gbm_weight_df = weight_df
#gbm_weight_df$koi_disposition = factor(weight_df$koi_disposition,labels=c("l1","l2"))
#set.seed(123)
#trainIndex <- createDataPartition(gbm_weight_df$koi_disposition, p = .8,
#                                  list = FALSE,
#                                  times = 1)

#training <- gbm_weight_df[ trainIndex,]
#testing  <- gbm_weight_df[-trainIndex,]

#fitControl <- trainControl(## 10-fold CV
 #                          method = "repeatedcv",
#                           number = 5,
#                           classProbs = TRUE,
#                           summaryFunction = twoClassSummary,
                           ## repeated ten times
#                           repeats = 1)
#gbmFit1 <- train(koi_disposition ~ ., data = training,
#                 metric = "ROC",
```

```
#                  method = "gbm",
#                  trControl = fitControl,
#                  verbose = FALSE)
```

```
#label_pred = predict(gbmFit1, testing)
#confusionMatrix(data = label_pred, testing$koi_disposition)
#p <- predict(gbmFit1,gbm_weight_model, type = "response")
```

```
x_test <- model.matrix(koi_disposition ~ .^2, data = sim_test)[,-1]
p = predict(weight_model, x_test, type = "response",s="lambda.min")
# classify observations based on a threshold
threshold <- 0.5

w_pre = p / (1-p)
w = length(w_pre)*w_pre/sum(w_pre)
```

```
# make predictions on sim_test data
x_test <- model.matrix(koi_disposition ~ ., data = sim_test)[,-1]
y_test <- sim_test$koi_disposition
pred_prob <- predict(simple_model, x_test, type = "response")

# This part is for the split-conformal type method discussed later
p_df = data.frame(class = as.double(y_test)-1, p = pred_prob)

p_df = p_df[order(-p_df$s0),]

max_i = 0

for (i in c(1:length(p_df$s0))) {
  subset <- p_df[c(1:i),]
  if (sum(subset$class * w[c(1:i)])/length(subset$class) < 0.9) {
    max_i = i
    break
  }
}

#print(p_df[max_i + 1,]$s0)

# classify observations based on a threshold
threshold <- 0.5
pred_class <- ifelse(pred_prob > threshold, 1, 0)

# create a confusion matrix and calculate accuracy
conf_mat <- table(y_test, pred_class)
accuracy <- sum(ifelse(pred_prob > threshold, 1, 0) == y_test)/length(y_test)
weighted_accuracy <- sum((ifelse(pred_prob > threshold, 1, 0) == y_test)*w)/length(y_test)
#conf_mat
#accuracy
#weighted_accuracy
```

Using this approach, the classifier trained to predict whether a point is labeled or unlabeled is able to achieve 77.71% prediction accuracy on this task. This again demonstrates the shift in covariate distribution between the labeled and unlabeled data; if the two come from the same distribution the classifier should be able to achieve predictive accuracy of no higher than around 50%. It is unlikely that this accuracy is achieved through overfitting since cross-validation was specifically used for this model to eliminate that issue. Note

that this accuracy is slightly different from the accuracy of a different model we fit for this task earlier in the paper due to slightly changes in model specification.

Using the weights produced from this model, the predicted accuracy for the confidence interval of predicting correctly on the unlabeled data (ie predicting a probability closer to the true value of the synthetic label than not) is estimated at 72.37%. This is a significant improvement on the 82.83% predicted accuracy estimated without weighting, but still significantly larger than the true accuracy of 67.87%. The reasons for this discrepancy are likely due to the ultimately low accuracy of the model used to produce the weights. Use of a stronger model for fitting the weights (gradient boosting) is also attempted, but the overall accuracy of the method to predict labeled versus unlabeled data only improved by 3% so the performance of the weights do not change substantially.

Matching Evaluation

The second method which is tested for its ability to account for distribution shift between the labeled and unlabeled data is the matching approach discussed previously. The same procedure as above is run but with the only originally labeled data used as the data selected by matching to be most like the originally unlabeled data. Before additional weighting, the matched data estimates an uncertainty of 78.33% while the actual accuracy on the synthetic data is 66.60%. The difference between estimated and observed uncertainty (~12%) for the matched data is significantly lower than before matching (~16%), but still does not seem to fully correct for the covariate shift. To bridge this additional gap, we apply weighting on top of the matched data, which ends up bringing the estimated accuracy down to 72.40% from the original 78.33%, but still a ways from the true value of ~66%. The parameters given are chosen to ensure that the models would converge and (in the case of the functions used to generate the synthetic labels) such that they have large coefficients for interaction terms. Regularization values of alpha = 0.05 and lambda = 0.1 are used when possible.

One-Side Split Conformal

The above analysis went in the opposite direction as typical conformal prediction - attempting to predict accuracy for a model on the unlabeled data given a set margin of error. Next we try to find a margin of error with $1 - \alpha$ coverage. To do this we use a modified version of split conformal prediction. In typical split conformal prediction we take our $\hat{u}$ predicted by our model and create a $1 - \alpha$ confidence interval around it using the $n + 1$ quantile for the residuals both above and below the prediction. In the context of binary classification, the actual $\hat{u}$ is always either 0 or 1 (the classification is either correct or incorrect). However we can configure the models we train to output a probability of one classification versus another. The difference between the actual values and the probabilities predicted by our model can form a type of residual which we can construct confidence intervals from. Here we attempt to construct a confidence interval in the form of the smallest threshold we can set for assigning a probability as a positive classification while still maintaining a $1 - \alpha$ accuracy rate. To do this, the originally labeled data is again separated into a training and testing set and used to try to predict the model's uncertainty on the originally unlabeled data. This method is successful if the threshold determined on the testing portion of the labeled data is higher than the threshold set by the unlabeled data because that means the coverage level determined on the labeled data is as or more conservative than necessary to achieve coverage on the unlabeled data.

Concretely, this threshold is determined by sorting the assigned probabilities on the testing portion of the labeled data from highest to lowest. Then we see how far we can go down that list of probabilities before less than 90% (for $\alpha = 0.1$) of the values predict a classification of 1 (that the object is an exoplanet). Our final threshold is the probability of the point right below the point at which we lost coverage (an adjustment analogous to taking the n+1 quantile for the regular split conformal procedure). Weights are applied in this procedure by instead of testing if the first i rows of the sorted probabilities achieve 90% accuracy we instead test if the first i rows achieve 90% weighted accuracy.

We next test if this approach achieves coverage on the held-out data once we include weights or once we only

use the matched labeled data.

Without weights, this approach sets a threshold of 0.976 using the labeled data while the unlabeled data only required a threshold of 0.687 for 90% accuracy. This does achieve coverage, but is enormously over conservative. With weights the threshold is set at 0.471 which no longer achieves coverage but is much closer to the true threshold than the unweighted approach.

In terms of matching, the matched data sets a threshold of 0.642 while the classifier trained on the matched data assigns probability 1 to one of the unlabeled points. Since there are fewer then 10 points assigned a probability of 1, this makes constructing a valid confidence interval for this model impossible since even setting a threshold of 1 leads to an accuracy of less than 90%.

Both matching and weighting decrease the estimated threshold. It is likely that both methods decrease the threshold since the points that the classification model are most certain about are the point most representative of the labeled distribution and thus dissimilar from the unlabeled distribution. However the purpose of adjusting the threshold to the unlabeled data is to make the threshold more conservative to account for the distribution shift. However we are not sure how to accomplish this.

Conclusions

Overall, our investigation involves three main parts. We first investigate the differences between the labeled and unlabeled data and verify that they display major differences in distribution. We then attempt two different approaches for accounting for this change in distribution (matching and weights) and use two different methods for calculating the uncertainty on synthetic labels for the unlabeled data for each. In terms of uncertainty estimated through accuracy, both the matching and weighting techniques improve the estimated accuracy of the model on the unlabeled data and work even better when combined. Both approaches tend to decrease the estimated threshold using the thresholding method but do not necessarily improve accuracy. This is likely due to the particularities of the specific fitted models used in estimation as well as the fact that the way the thresholding method is set up both the weights and matching techniques end up making the thresholds less conservative, not more. This would be a rewarding area to look into in further work.

References

1. Tibshirani, Ryan J., Rina Foygel Barber, Emmanuel Candes, and Aaditya Ramdas. "Conformal prediction under covariate shift." Advances in neural information processing systems 32 (2019).

Supplemental Code for Matching Evaluation

```
koi_df = read.csv("KOI.csv")

# Get the unlabeled data
unlabeled_df = koi_df[c("ra","dec","koi_period","koi_time0bk","koi_impact",
                        "koi_duration","koi_depth","koi_prad","koi_teq",
                        "koi_insol","koi_steff","koi_slogg","koi_srad",
                        "koi_disposition")][koi_df$koi_disposition == "CANDIDATE",]
unlabeled_df = unlabeled_df[complete.cases(unlabeled_df), ]
```

```
# Get the matched data
analysis_df = read.csv("matched_data_nearest.csv")
analysis_df = analysis_df[c("ra","dec","koi_period","koi_time0bk","koi_impact",
                            "koi_duration","koi_depth","koi_prad","koi_teq",
                            "koi_insol","koi_steff","koi_slogg","koi_srad",
                            "koi_disposition")][koi_df$koi_disposition != "CANDIDATE",]
analysis_df = analysis_df[complete.cases(analysis_df), ]
analysis_df$koi_disposition = as.factor(analysis_df$koi_disposition)
analysis_df$koi_disposition <- factor(analysis_df$koi_disposition, labels = c(1,0))

# Train complex classifier on half_analysis
half_analysis = analysis_df[1:400,]
# Predict the labels on the other half of the labeled data h2_analysis
h2_analysis = analysis_df[-c(1:400),]
```

```
x <- model.matrix(koi_disposition ~ .^2 + I(log(koi_period)) + I(log(koi_duration)),data = half_analysis
y <- half_analysis$koi_disposition
complex_model <- glmnet(x, y, family = "binomial", alpha = 0.001, lambda = 0.0001)
```

```
#coef(complex_model, s = 0.1)
```

```
x_pred <- model.matrix(koi_disposition ~ .^2+ I(log(koi_period)) + I(log(koi_impact)),
                       data = h2_analysis)[,-1]
h2_analysis$koi_disposition = factor(as.vector(predict(complex_model,x_pred)) > 0.5,
                                     labels = c(0,1))
x_pred <- model.matrix(koi_disposition ~ .^2+ I(log(koi_period)) + I(log(koi_impact)),
                       data = unlabeled_df)[,-1]
unlabeled_df$koi_disposition = factor(as.vector(predict(complex_model,x_pred)) > 0.5,
                                      labels=c(0,1))
```

```
sim_train = h2_analysis[1:700,]
sim_test = h2_analysis[-c(1:700),]
x <- model.matrix(koi_disposition ~ ., data = sim_train)[,-1] # remove intercept column
y <- sim_train$koi_disposition
simple_model <- glmnet(x, y, family = "binomial", alpha = 0.05, lambda = 0.1)
```

```
# make predictions on sim_test data
x_test <- model.matrix(koi_disposition ~ ., data = sim_test)[,-1]
y_test <- sim_test$koi_disposition
pred_prob <- predict(simple_model, x_test, type = "response")

# This part is for the split-conformal type method discussed later
p_df = data.frame(class = as.double(y_test)-1, p = pred_prob)
```

```r
p_df = p_df[order(-p_df$s0),]

max_i = 0

for (i in c(1:length(p_df$s0))) {
  subset <- p_df[c(1:i),]
  if (sum(subset$class)/length(subset$class) < 0.9) {
    max_i = i
    break
  }
}

#print(p_df[max_i + 1,]$s0)

# classify observations based on a threshold
threshold <- 0.5
pred_class <- ifelse(pred_prob > threshold, 1, 0)

# create a confusion matrix and calculate accuracy
conf_mat <- table(y_test, pred_class)
accuracy <- sum(diag(conf_mat)) / sum(conf_mat)
#conf_mat
#accuracy
```

```r
# make predictions on sim_test data
x_test <- model.matrix(koi_disposition ~ ., data = unlabeled_df)[,-1]
y_test <- unlabeled_df$koi_disposition
pred_prob <- predict(simple_model, x_test, type = "response")

# This part is for the split-conformal type method discussed later
p_df = data.frame(class = as.double(y_test)-1, p = pred_prob)

p_df = p_df[order(-p_df$s0),]

max_i = 0

for (i in c(1:length(p_df$s0))) {
  subset <- p_df[c(1:i),]
  if (sum(subset$class)/length(subset$class) < 0.9) {
    max_i = i
    break
  }
}

#print(p_df[max_i + 1,]$s0)

# classify observations based on a threshold
threshold <- 0.5
pred_class <- ifelse(pred_prob > threshold, 1, 0)

# create a confusion matrix and calculate accuracy
conf_mat <- table(y_test, pred_class)
accuracy <- sum(diag(conf_mat)) / sum(conf_mat)
#conf_mat
```

```r
#accuracy
```

```r
weight_1 <- sim_test
weight_2 <- unlabeled_df

weight_1$koi_disposition = 0
weight_2$koi_disposition = 1
# Matching the size of originally labeled and originally unlabeled data in the training data
weight_df = rbind(weight_1, weight_2[1:500,])
weight_df$koi_disposition = factor(weight_df$koi_disposition)
table(weight_df$koi_disposition)
```

```
##
##   0   1
## 660 500
```

```r
# Fit the model with cross-validation
# create model matrix and outcome vector
x <- model.matrix(koi_disposition ~ .^2, data = weight_df)[,-1] # remove intercept column
y <- weight_df$koi_disposition

# perform cross-validation with cv.glmnet
cvfit <- cv.glmnet(x, y, family = "binomial", alpha = 0.1,nfolds=5)

# get predicted probabilities and classes for each fold using predict function
p <- predict(cvfit, x, type = "response", s = "lambda.min")
# use lambda that gives minimum mean cross-validated error
threshold <- 0.5 # set classification threshold
pred_class <- ifelse(p > threshold, 1, 0) # convert probabilities to classes

# create a confusion matrix and calculate accuracy for each fold and overall
conf_mat <- table(y, pred_class) # confusion matrix
accuracy <- sum(diag(conf_mat)) / sum(conf_mat) # overall accuracy

# print results
#print(conf_mat)
#print(accuracy)
```

```r
x_test <- model.matrix(koi_disposition ~ .^2, data = sim_test)[,-1]
p = predict(cvfit, x_test, type = "response")
# classify observations based on a threshold
threshold <- 0.5

w_pre = p / (1-p)
w = length(w_pre)*w_pre/sum(w_pre)
```

```r
# make predictions on sim_test data
x_test <- model.matrix(koi_disposition ~ ., data = sim_test)[,-1]
y_test <- sim_test$koi_disposition
pred_prob <- predict(simple_model, x_test, type = "response")

# This part is for the split-conformal type method discussed later
p_df = data.frame(class = as.double(y_test)-1, p = pred_prob)

p_df = p_df[order(-p_df$s0),]
```

```r
max_i = 0

for (i in c(1:length(p_df$s0))) {
  subset <- p_df[c(1:i),]
  if (sum(subset$class*w[c(1:i)])/length(subset$class) < 0.9) {
    max_i = i
    break
  }
}

#print(p_df[max_i + 1,]$s0)

# classify observations based on a threshold
threshold <- 0.5
pred_class <- ifelse(pred_prob > threshold, 1, 0)

# create a confusion matrix and calculate accuracy
conf_mat <- table(y_test, pred_class)
accuracy <- sum(ifelse(pred_prob > threshold, 1, 0) == y_test)/length(y_test)
weighted_accuracy <- sum((ifelse(pred_prob > threshold, 1, 0) == y_test)*w)/length(y_test)
#conf_mat
#accuracy
#weighted_accuracy
```

1 Project 2 **78 / 100**

Questions, ideas, & design of analysis (40pts) [Data option only]

* The project is designed in a creative and thoughtful way, to address interesting questions and challenges in the data

* The real world meaning of the data is considered in an insightful way to guide the design of the analysis

* Choices made along the way, for example designing a test statistic or finding a way to measure or visualize results, are addressed thoughtfully

* The analysis shows thorough understanding of any preexisting tools, code, packages, etc, that the group chose to use, and these choices are well suited to the problem at hand

+ 37 pts Click here to replace this description.

✓ **+ 34 pts** *Click here to replace this description.*

Statistical methodology (30pts) [Data option only]

* The analysis identifies all the major relevant challenges, such as issues of multiple testing, non-independence, confounding, exploratory data analysis, etc

* These challenges are handled appropriately in the analysis, applying existing tools or developing new techniques

* Assumptions are tested and examined using, e.g., using diagnostic plots

* Any remaining issues that cannot be addressed, are discussed

+ 26 pts Click here to replace this description.

✓ **+ 22 pts** *Click here to replace this description.*

Statistical methods / ideas / simulation design (70pts) [Theory option only]

* The project offers an insightful overview of the paper, with a well presented summary and with thoughtful comments and/or critique on the paper

* The report poses interesting and deep questions regarding extensions or limitations of the paper

* (If applicable) The new methods proposed in the project are well motivated and thoughtfully constructed

* (If applicable) The new theoretical results developed in the project are correct, are clearly presented and proved, and are justified in terms of why they are meaningful to the

methodology

* (If applicable) The simulations or experiments in the project are designed in a thoughtful and interesting way to address important questions about the properties or limitations of the methods being tested, and their results are recorded and interpreted well

* Any remaining issues that cannot be addressed, are discussed

+ 66 pts Click here to replace this description.

+ 56 pts Click here to replace this description.

+ 64 pts Click here to replace this description.

+ 55 pts Click here to replace this description.

+ 65 pts Click here to replace this description.

+ 50 pts Click here to replace this description.

+ 58 pts Click here to replace this description.

+ 54 pts Click here to replace this description.

Report & code (30pts)

* The report is clear and well-written, presenting a cohesive and well motivated explanation of the path followed in the analysis, and thoughtful and justified conclusions based on the findings

* Open questions, uncertainties due to insufficient data, questions relating to untestable assumptions, etc, are addressed as needed

* The code is clear, well organized, and appears readable and reproducible

* Sufficient details are given to understand the specifics of the analysis being run and the choices made along the way

+ 27 pts Click here to replace this description.

✓ **+ 22 pts** *Click here to replace this description.*

+ 29 pts Click here to replace this description.

+ 24 pts Click here to replace this description.

+ 25 pts Click here to replace this description.

+ 28 pts Click here to replace this description.

+ 26 pts Click here to replace this description.

Nice work on your project!

Comments on "Questions, ideas, & design of analysis":

Your overall approach is to train a gradient boosting model on a training set, then use a carefully constructed holdout set to estimate accuracy on the test data. The holdout set is constructed by matching data points from the test set, essentially to choose a "control group" point for each data point in the test set. The estimated accuracy is, as expected, lower (and likely more reliable) after this matching process as compared to using the "raw" holdout set without matching.
(As an aside, you chose gradient boosting because logistic seemed to have low accuracy, <80% , in your initial experiment -- however in my code I think logistic has >95% so I'm not sure exactly how this happened).
Overall, this is a solid approach and clearly the matching shows a lot of improvement in the covariate distribution. The synthetic data experiment for verifying the improvement in uncertainty quantification has some issues (see more comments below), and it would be good to explore how the matching procedure affects the results / different options for matching / what is the best way to measure distance (more comments on "matchit" below).

Comments on "Statistical methodology":
The main process for ensuring validity is the matching approach for creating a holdout set that is more similar to the test set. This is a very solid approach to the problem of covariate shift in this data set, and the visualizations and numerical results showing the improvement in the similarity are very compelling. One issue is that the matching is carried out with the "matchit" function and it's not fully discussed what exactly is being done by this function -- for example, since the different variables are on very different scales, what is the meaning of "nearest" / of distance, in this context? In addition there may be some outliers that should be discarded (perhaps this is the reason that even after matching, some variables like koi_period are still very different).

A more important issue is that the simulation carried out with the synthetic dataset (which is intended for checking if the matched-holdout procedure is indeed accomplishing the desired result) is carried out with multiple different approaches, only some of which are targeted towards this intended goal. Specifically, the way we would want to do this is:
* Create a synthetic dataset where the training set and the test set exhibit some covariate shift . This is what your code does successfully.
* Then, run *the same method(s)* you run on the real data, on this synthetic data, to see how it performs. This appears to be what you do in the "Matching Evaluations" subsection described on page 27, but the "Weighting Evaluation" (i.e., weighted conformal) described before, and the "One side split conformal" described after, appear to be separate from any methods you ran on the real

data.

Comments on "Report & code":
The report covers a good level of detail and the code is well integrated into the report. The visualizations are excellent. One issue is that of clarity in terms of presenting the high level ideas. It's clear from the report that your group's initial way of thinking about the problem was by way of conformal prediction, and then you moved away from this due to the binary label data; this is the path you followed towards designing your final procedure, but it's not the most clear way to present the work because you are centering your discussion around a methodology (i.e., conformal) that doesn't arise in your project [except in the simulation at the end -- see comments above]. For example, on page 1 it would be more clear to give an overview of what route you took rather than describing why you didn't use conformal. Similarly on page 21 (the "Weighted Accuracy" section) you can simply say that the fraction correct in the holdout set estimates the fraction correct in the test set (where prediction is 1 if fitted probability is >0.5, or 0 otherwise), without discussing margin of error / conformal / etc. One other issue as mentioned earlier is that using a preexisting package ("matchit") should be accompanied by a more clear explanation of exactly what process the function is performing.