"Predictive Model for Culex Mosquito Density in Metro Philadelphia"

Britton Hartzok

4/21/2021

Introduction

In epidemiology, vectors refer to a diverse range of organisms that act as a bridge between pathogen and host species. Worldwide, nearly 20% of all human deaths are caused by illnesses classified as vector-borne such as malaria, dengue, and yellow fever (Control., 2014). Although mosquitoes, ticks, and rodents are the most prevalent carriers of disease, any organism has the propensity to be classified as a vector - such as copepods, bats, biting flies, fleas, or even rabid canines. This study specifically seeks to explore the primary bridge of West Nile virus to humans- Culex pipiens-restuans mosquitoes.

West Nile virus (WNV) was first detected on the North American continent in 1999 in New York state (Nash, et al., 2001). At a 96% pool positivity rate, Culex-genus mosquitoes (tarsalis, quinquefasciatus, pipiens, restuans) are the primary transmitters of WNV (Andreadis, 2012). Cx. pipiens and restuans are endemic to the Mid-Atlantic US, and therefore are monitored heavily within the region. Although WNV has its own virologic cycle, tracking the Culex-genus provides more predictability and enhances seasonal vector management strategies.

Although distinct taxonomically, Cx. pipiens and restuans have a strikingly similar morphology making differentiation between the two very time consuming and prone to misidentification - sometimes due to loss of key phenotypical features during handling (Harrington & Poulson, 2008). Because of shared breeding habitats and high WNV positivity rates, Cx. pipiens-restuans are often collected and tested together in the same pool. For this reason, no distinguishment will be made between the them in this study.

Culex mosquitoes go through four life stages - three aquatic (egg, larva, pupa) and the last being the terrestrial, airborne adult (Karki, et al., 2016). Shortly after taking a blood meal from a host, pregnant females begin to seek out suitable habitat for oviposition - standing water with adequate levels of organic matter for larvae to feed (Karki, et al., 2016). Air temperature as well as water temperature of the larval habitat are a main drivers of the mosquito biological cycle (Reisen, et al., 2006). Dictated by weather and other environmental cues, eggs will hatch to larvae and emerge as adults in 10 to 14 days (Crans, 2004). Imagos mosquitoes are greatly impacted by warmer air temperature by shortening the resting period between blood meal and oviposition and encouraging swarming behavior (Reisen, et al., 2006). In addition, temperature has been known to influence senescence. Andreadis, et al. (2014) found shorter life spans at 30^{o}C and above, while greatest longevity was observed between 15 and 30^{o}C. In temperate climates, like Pennsylvania, female Culex populations persist through the fall until daylight hours and overnight temperatures initiate diapause - their overwintering mechanism (Spielman & Wong, 1973).

Other than temperature, major cyclical drivers of mosquito abundance include relative humidity, precipitation, daytime length, wind speed, and seasonally available aquatic habitats. Preferred breeding sites rely on the presence of stagnant water most often including wetlands, shallow bodies of water, catchment and retention basins, sewage treatment plants, and artificial containers - i.e. tires, vacant swimming pools, dirty bird baths, clogged rain gutters, and vacant ornamental ponds. It is natural to conclude that rain correlates

to mosquito activity. But in some cases, precipitation can temporarily dampen mosquito presence. Low-intensity rainfall might create or replenish breeding habitat, whereas heavy rain events could result in larval flushing thereby limiting emergence of future generations (Jones, et al., 2012).

Culex population density is the main regulator of local West Nile virus infections, and public health decisions are largely influenced by local vector observations (Karki, et al., 2016). Local protocols and control strategies must mirror surveillance efforts. The purpose of this study is to identify which weather parameters are significant predictors of Culex pipiens-restuans abundance. The area of interest covers populated regions of Pennsylvania and relies upon 5 years of state-provided Culex collection data and surface weather observations at local airports. Due to the size and inconsistency of the data, only annually monitored trapping sites were sampled. In this analysis, some observations were smoothed or normalized to create a standard by which a reasonable conclusion could be made. In some cases this approach was helpful. In other cases, highly variable observations (such as precipitation) could have been smoothed to such an extent that strength of correlation was lost. The general correlations found within this report could assist local vector managers to create a local, short-term predictive model that might help anticipate control measures.

Methodology

My project focuses specifically on Philadelphia because of its manageable size and the abundance of data. In past work, I tried to model the entire state and found that so much smoothing and generalizing weakened relationships between variables. In addition, the regions were so big that in some cases I was relying on airport weather data from many miles away and had no way of controlling for variability. In addition, I relied heavily on temperature metrics in my past project, however this was really limiting since all of my temperature variables were mutually dependent and collinear (i.e humidity, dewpoint, heat index). I've included stream stage data in this model because I think it could be an important independent variable that might capture the effects of short-term precipitation over an entire basin (rather than at just one recording station) as well as longer term soil moisture and water table level.

All of my mosquito collection data was provided by the PA Department of Environmental Protection West Nile monitoring program. Although this dataset runs from 2015-2020, I only used 2017-2020 because of limited stream data from years 2015-2016. In the database provided by the DEP, I extracted just Delaware, Chester, Bucks, Montgomery, and Philadelphia counties. Then, I filtered these observations based on locations that were monitored weekly from 2017-2020 which shrunk the data down to just a few hundred sites. After that, I chose 56 trapping locations that produced the greatest amount of mosquitoes - based on cumulative and mean values. I took this step as a way of controlling for unique site characteristics, and their various roles in determining mosquito density. Heavily urban areas have very few mosquito breeding habitats aside from sewer catch basins. Lighter urban settings have more (abandoned) swimming pools and littered backyards. While, suburban areas (especially those developed in the past 10-20 years) have large retention basins and ponds that have the potential to become major breeding grounds for female mosquitoes especially when there are no established predators such as fish and amphibians. Therefore, I filtered my data selection towards habitats that are sensitive to pertubations in the weather. The map below provides some spatial awareness of the area I'm studying including all mosquito monitoring sites, airports, and the Darby-Cobbs Creek watershed.

For the weather data, I utilized the Iowa State Environmental Mesonet which has a historical catalog of airport METAR weather data. The four airports I selected were Philadelphia International (PHL), Philadelphia Northeast (PNE), Brandywine Regional (OQN), and Wings Field (LOM), and I chose these sites because they are roughly located on the four corners of my area of interest. Furthermore, elevation changes in my study vary from close to sea-level near the Delaware River to about 300 feet further up in Mainline Philadelphia. I could have made simple temperature-elevation adjustments based on observation at the PHL airport, but I didn't feel this fully captured all the microclimates. On the opposite side, I considered employing something like the Thiessen Polygon method which would have allowed for temperature corrections at an acceptable level of accuracy. However, this approach would have been prohibitively time-consuming in the scope of my project. So, I simply chose to average all observations together into one daily mean that

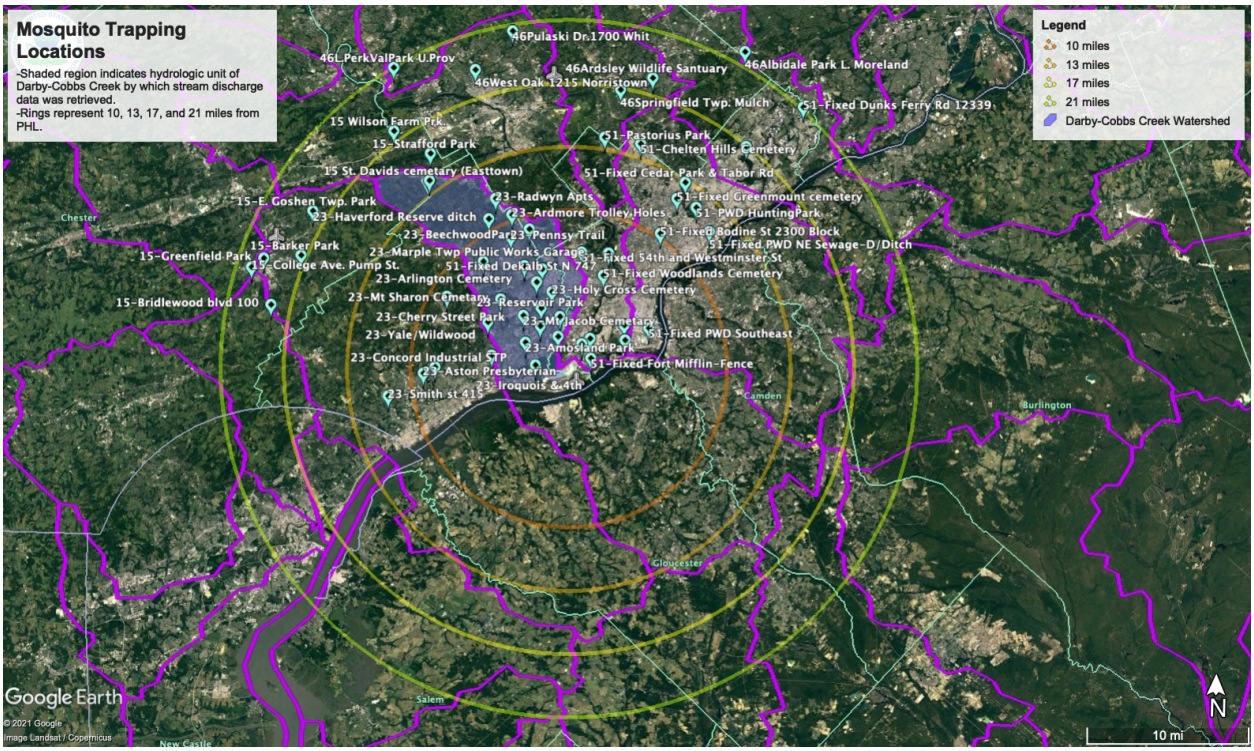


Figure 1: **Created in Google Earth Pro**

I'm considering as the daily weather over metro Philadelphia on a given day. It is also worth noting that PNE, OQN, and LOM only reliably contributed temperature data to this project. Whereas, data from PHL included temperature, humidity, dewpoint, and sky conditions.

When considering stream gage data, I needed to include a watershed and stream that doesn't drain outside of my area of interest so that I can more accurately generalize soil moisture, precipitation, and the water table. As seen in the map above, the combined Darby-Cobbs Creek drains the central portion of Philadelphia and is entirely contained within the area of study. Furthermore, I needed a stream monitoring station located at the mouth of a watershed so as to accurately capture total rainfall within that hydrologic unit. The US Geologic Survey maintains a network of stream monitoring stations in the area, and I was able retrieve Darby-Cobbs Creek flow data from the 84th Street Bridge site. As previously mentioned, data was extremely sparse from 2015-2016 (perhaps due to technical reasons), so I chose just to examine 2017-2020.

Variable Selection

Weather (AWOS/METAR)

-Month (integer)
-Week number elapsed from Jan 1st (integer)
-Min/Mean/Max Temp (°F)
-DD (Base 50°F) = Average Daily Temp - 50°. If average temp is less than 50°, DD = 0.
-Min/Max Dewpoints (°F)
-Max Relative Humidity (%)
-Mean Windspeed (kts)

-Min Visibility (mi)
-Average Daily Sky Cover at Lowest Measured Elevation

- Average Daily Sky Cover consists of 8 variables that have been grouped together into 3 classes and given binary identifiers (1-present, 0-absent). PHL monitors sky cover every 15 minutes, so I'm creating a daily average here by coming up with decimal values that will represent frequency of these conditions:
  1. **Clear** = Scattered Sun, No Clouds, Clear
  2. **Scattered** = Scattered Clouds, Few Clouds, Broken Clouds
  3. **Overcast** = Overcast, Vertical Visibility (dense fog or heavy precipitation)

From personal experience and having read journals on past research, I'm also going to incorporate a 2-point smooth to all variables. That is, I will average the previous and current day's observation into one value which will be more representative of the current day by better capturing night-time conditions. For example, dates are usually interpreted as beginning at 12:00AM and ending at 11:59PM, and I need to account for the nighttime hours prior to 12:00AM. Given the characteristics of certain frontal systems, I can't say for certain when a low or high temperature will occur in a day. A cold front might enter when we'd usually expect a high temperature and a warm front will sometimes move in during the evening hours when we'd expect our low temperature. In addition, mosquito traps were set on the previous day and the collection date is indicated as the day when the traps is collected. My approach in normalizing the data should preserve the expected relationships.

Finally, I'd like to experiment with the cumulative relationship of these parameters to mosquito density using various rolling averages from the previous 3 to 20 days. Variables that greatly influence night time swarming behavior (such as sky cover and windspeed) will largely be examined on short-term intervals. However, stream gage, which could influence the speed of the greater mosquito life cycle, would be more appropriately analyzed in the 10-20 days prior.

Stream Gage Height (USGS Stream Monitoring Network)

The data set from Darby-Cobbs Creek was to include gage height, discharge, water quality, and water temperature. Upon retrieval of the data, discharge, water quality, and temperature were largely missing to the point where imputation would be highly unreliable which is disappointing because water temperature would have been extremely useful. Because discharge and gage are directly related and because I'm really only interested in changes to stream flow and not the values themselves, gage height should be a suitable parameter.

Parameters:
- Daily Min/Mean/Max Stream Gage Height

Mosquito Collection Data (PA DEP West Nile Program)

As you might expect, mosquito counts can be highly variable from location to location and from night to night. In addition, I have no information about adulticide/larvicide treatments near these sites in the time leading up to capture. I'm uncertain of the degree to which this complicates the analysis. In the past, I've also experimented with trimmed means and cumulative counts which have returned questionable results. Because my mosquito data is highly skewed, I'm going to rely on R packages to perform scaling and transformation where needed so that my variables can be compared on the same level. Given the complexity of modeling biological processess and my previously underwhelming attempts at performing linear regression, I'm going to employ machine learning classification techniques that can better cut through the noise of my data. Finally, just as I had averaged all weather observations from our 4 airports into one daily mean, I will do the same for mosquito collection data as well.

Simple Multivariate Linear Regression vs. Machine Learning Techniques

Below, I'll present some very basic results of a linear regression just as means of contrasting results found using more advanced machine learning algorithms.

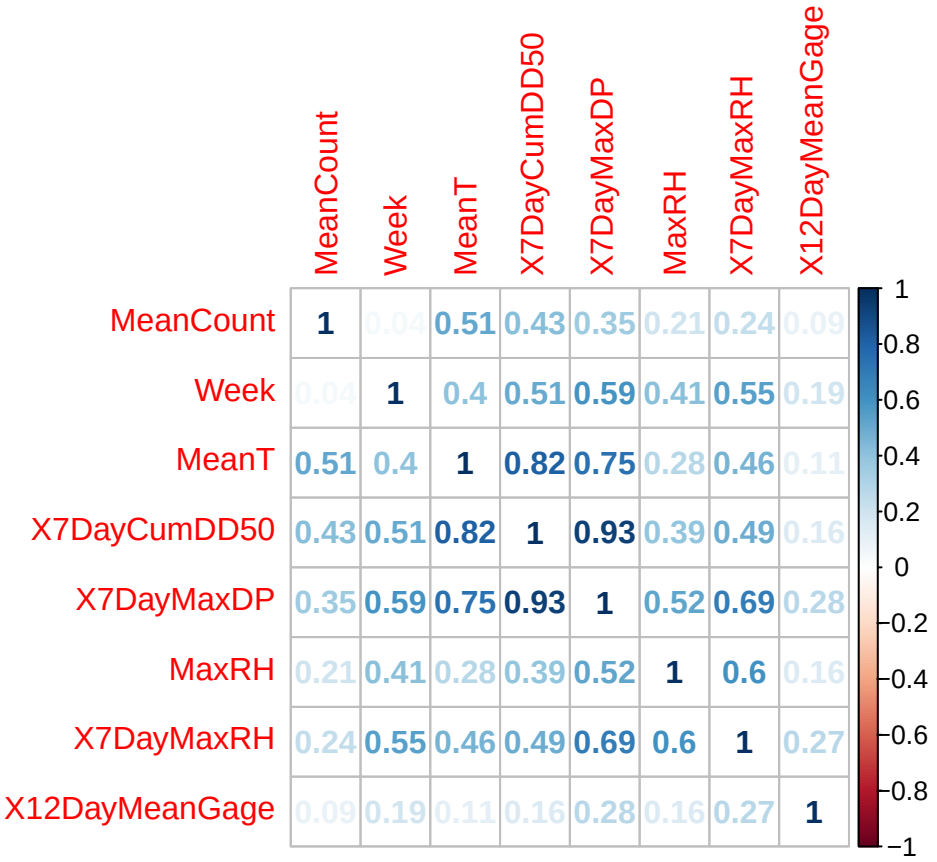
Linear Model

```
knitr::opts_chunk$set(fig.width=12, fig.height=8)
#Linear Regression
mos<- read.csv("mosq.csv", header=T)
#Drop date column
mos$Date<-NULL
#Scale and center data on -1,1 range
preProcValues <- preProcess(mos, method = c("center", "scale"))
mos <- predict(preProcValues, mos)
set.seed(13)
#Linear Model
lm.all <- lm(MeanCount~., data=mos)
#Variables with p-value > 0.05
mos<-mos[,-c(4,7,8,9,10,13,14,15,17,22,
             25,27,29,30,33,34,35,36,38,39,44,43,42,21)]
lm.tune <-lm(MeanCount~., data=mos)
#More trimming of variables > 0.05
mos<-mos[,-c(3,4,7,8,12,14,16,19,20)]
lm.tune1 <-lm(MeanCount~., data=mos)
#...Again
mos<-mos[,-c(11,9)]
lm.tune2 <-lm(MeanCount~., data=mos)
#...And Again
mos<-mos[,-c(8)]
lm.tune3 <-lm(MeanCount~., data=mos)
#Final Model
summary(lm.tune3)
```

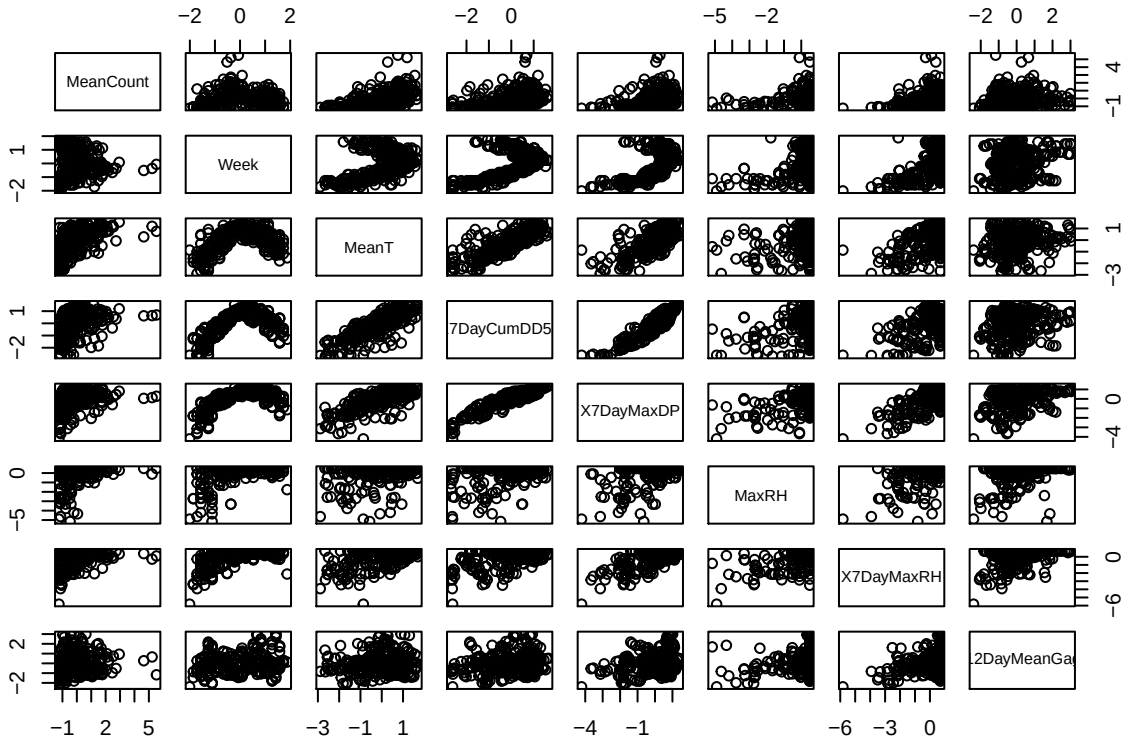
```
##
## Call:
## lm(formula = MeanCount ~ ., data = mos)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.8605 -0.5387 -0.0720  0.3124  4.9260
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -3.048e-16  4.528e-02   0.000  1.00000
## Week           -2.545e-01  5.811e-02  -4.380 1.62e-05 ***
## MeanT           4.109e-01  8.295e-02   4.953 1.19e-06 ***
## X7DayCumDD50    7.707e-01  1.889e-01   4.080 5.70e-05 ***
## X7DayMaxDP     -7.992e-01  2.016e-01  -3.965 9.08e-05 ***
## MaxRH           1.461e-01  5.878e-02   2.486  0.01342 *
```

```
## X7DayMaxRH      2.438e-01  8.474e-02   2.877  0.00429 **
## X12DayMeanGage  1.098e-01  4.966e-02   2.212  0.02767 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8163 on 317 degrees of freedom
## Multiple R-squared:  0.348,  Adjusted R-squared:  0.3336
## F-statistic: 24.17 on 7 and 317 DF,  p-value: < 2.2e-16
```

```
mos.cor<-cor(mos)
#Correlation matrix plot with Pearson r-coefficients listed
corrplot(mos.cor, method='number')
```




```
#Final variables plotted.  Non-linearity very apparent
plot(mos)
```



After trimming insignificant variables and with a final r-squared value of 0.348, this linear model wouldn't be a particularly accurate predictor of mosquito catch even when applied to a test set.

Machine Learning Techniques

Support Vector Machine (SVM) for Numerical Prediction

Although not often employed in this way, SVMs can also make pretty reasonable numerical predictions. Here I'll feed and tune a training set into various SVMs using linear, radial, and polynomial kernals and compare predictions to a test set.

Re-Reading Data

```
#re-reading the data table
mos<- read.csv("mosq.csv", header=T)
#Date Null
mos$Date<-NULL
#Data set is entirely numerical
# str(mos)
#Scaling from -1,1
preProcValues <- preProcess(mos, method = c("center", "scale"))
mos <- predict(preProcValues, mos)
```

Training/Test Set Creation

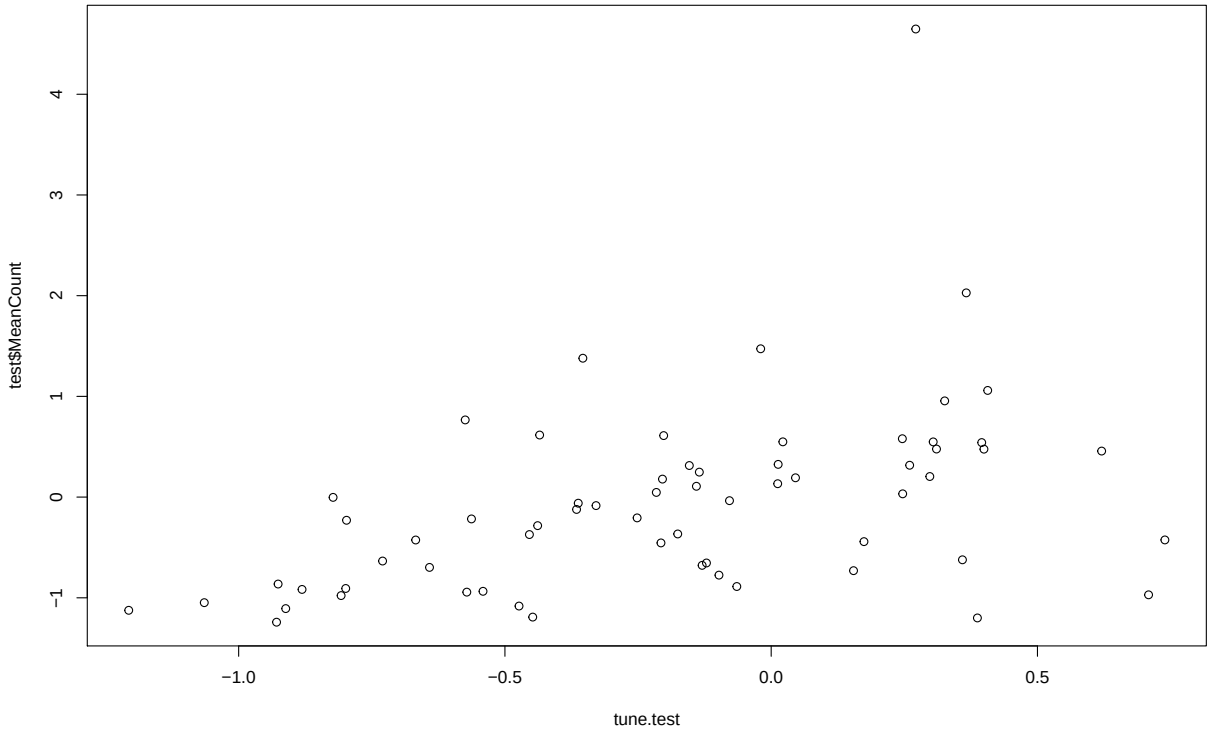
```
set.seed(1981)
#Training/test set at 85:15 ratio
trainIndex <- createDataPartition(mos$MeanCount,
                                  p = 0.8, list = FALSE, times = 1)
#Parse out training/test sets
train <- mos[ trainIndex,]
test <- mos[-trainIndex,]
```

Linear Kernal

```
set.seed(311)
linear.tune<-tune.svm(MeanCount~.,data=train,kernel="linear",
                      cost = c(.001,.01,.1,1,5,10,100))
summary(linear.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.729939
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.8143566  0.4597445
## 2 1e-02 0.7299390  0.3980178
## 3 1e-01 0.7364341  0.3484378
## 4 1e+00 0.8227410  0.2938785
## 5 5e+00 0.8590358  0.2806036
## 6 1e+01 0.8671853  0.2856287
## 7 1e+02 0.8745010  0.2949238
```

```
best.linear<-linear.tune$best.model
tune.test<-predict(best.linear,newdata=test)
plot(tune.test,test$MeanCount)
```




```
tune.test.resid<-tune.test-test$MeanCount
#Mean Squared Error
mean(tune.test.resid^2)
```

```
## [1] 0.7244752
```

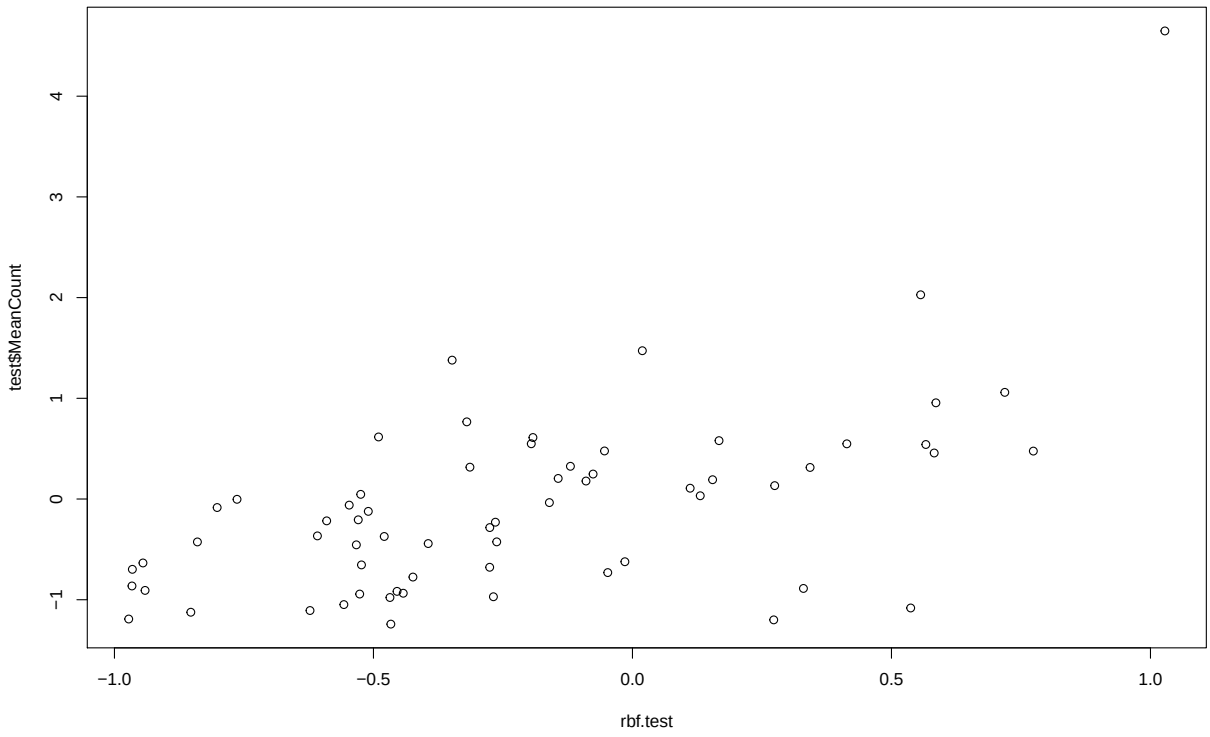
RBF Kernal

```
set.seed(1313)
rbf.tune<-tune.svm(MeanCount~.,data=train,kernel="radial",
                   gamma = c(.1,.5,1,2,3,4),
                   cost = c(.001,.01,.1,1,5,10,100))
rbf.tune$best.model
```

```
##
## Call:
## best.svm(x = MeanCount ~ ., data = train, gamma = c(0.1, 0.5, 1,
##     2, 3, 4), cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
```

```
##       gamma:  0.1
##     epsilon:  0.1
##
##
## Number of Support Vectors:  226
```

```
best.rbf<-rbf.tune$best.model
rbf.test<-predict(best.rbf,newdata=test)
plot(rbf.test,test$MeanCount)
```



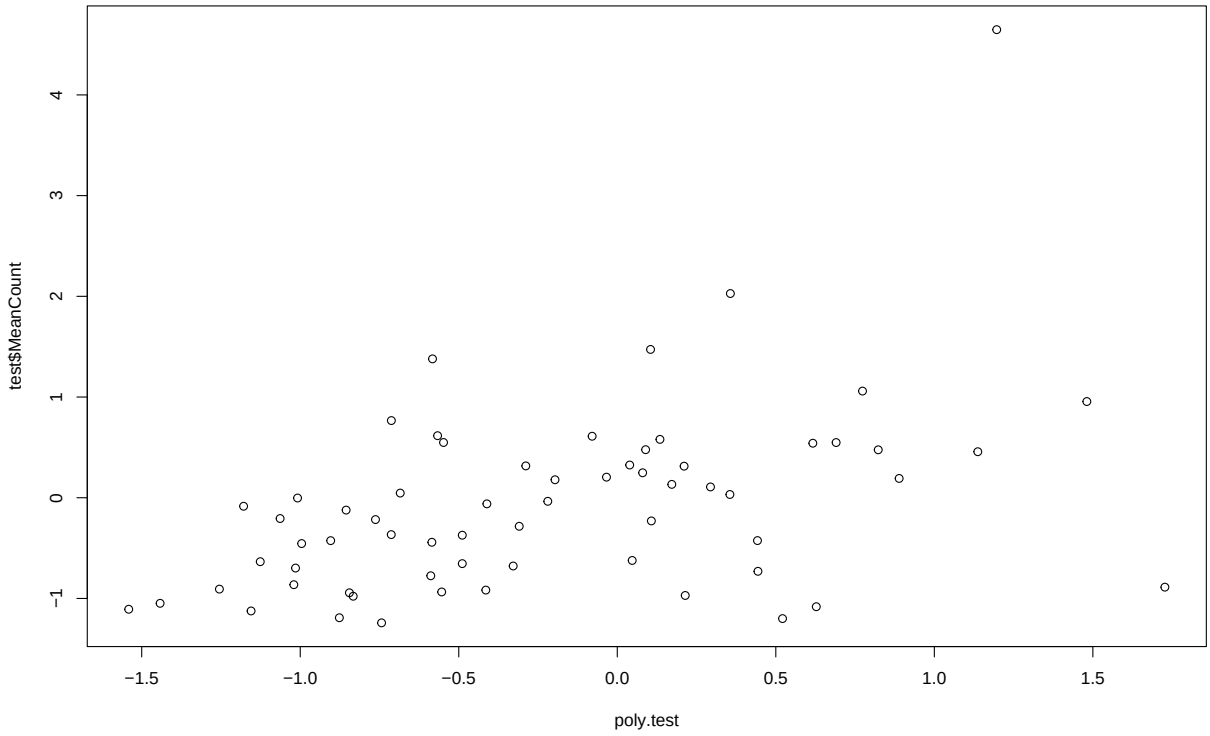

```
rbf.test.resid<-rbf.test-test$MeanCount
#Mean Squared Error
mean(rbf.test.resid^2)
```

```
## [1] 0.5974542
```

Poly Kernal

```
knitr::opts_chunk$set(fig.width=12, fig.height=8)
set.seed(123)
poly.tune<-tune.svm(MeanCount~.,data = train,kernal="polynomial",
                    degree = c(1,2,3,4,5),coef0 = c(.1,.5,1,2,3,4),
                    cost = c(.001,.01,.1,1,5,10,100))
best.poly<-poly.tune$best.model
```

```
poly.test<-predict(best.poly,newdata=test)
plot(poly.test,test$MeanCount)
```




```
poly.test.resid<-poly.test-test$MeanCount
mean(poly.test.resid^2)
```

```
## [1] 0.81361
```

Going off of MSE, the linear and radial kernals performed highest on the unseen test set compared to the polynomial. However, at a mean squared error of around 0.45, improvements can be made by testing other algorithms or simply by converting our response variable (MeanCount) into two or more classes and reapplying our techniques. Taking a step back, what is the purpose of creating a model that reliably predicts average trap counts down to a single mosquito? Given the amount of immeasurable predictor variables and the complexity of this problem, numerical models could be overly challenging and unrealistic in their application. From a vector management standpoint, it might actually be more useful to know when mosquito pressure will fall into a certain range or when it is expected to exceed a certain threshold. In the following steps, I'm going to transform MeanCount into both a binary and multiclass class variable and then reapply the methods employed above as well as new techniques that are available to us now that we're approaching this problem from a classification perspective.

Binary Classification

Having minimal experience in the field of vector management, I can only make assumptions. However, here I'd like to establish a hypothetical threshold for the mean daily mosquito count. Any MeanCount at or

exceeding 100 will be classified as "Hi" and anything below that threshold will be classified as "Lo". A mean count of 100 or greater could be concerning to a vector control professional and may warrant changes to operational strategy or could serve as an early warning system. Assuming a class imbalance of an unknown degree will exist, I'm going to test our models on imbalanced training/test sets and then perhaps address this imbalance through synthetic sampling techniques to see if improvements can be made.

Data Read-In, Preprocessing, Training

```
mos<- read.csv("mosq.csv", header=T)
mos$Date<-NULL
mos$Week<-as.factor(mos$Week)
mos$Month<-as.factor(mos$Month)
#Rounding Mean Counts to 1 significant figure which makes for easy factoring
mos$MeanCount<-signif(mos$MeanCount, digits=1)
#With rounding, if MeanCount is less than 100, that observation will be classified as "Lo",
#all others will be classified as "Hi".
mos$MeanCount <- ifelse(mos$MeanCount < 100, "Lo","Hi")
#Convert MeanCount to a factor with 2 levels
mos$MeanCount<-as.factor(mos$MeanCount)
#Scale and center all numerical values
preProcValues <- preProcess(mos, method = c("center", "scale"))
mos <- predict(preProcValues, mos)
#Training and test set creation (85:15)
set.seed(45)
trainIndex <- createDataPartition(mos$MeanCount, p = 0.85, list = FALSE, times = 1)
#Parse out training/test sets
train <- mos[ trainIndex,]
test <- mos[-trainIndex,]
#Check class distribution (slightly imbalanced)
prop.table(table(train$MeanCount))
```

```
##
##       Hi       Lo
## 0.299639 0.700361
```

```
prop.table(table(test$MeanCount))
```

```
##
##        Hi        Lo
## 0.2916667 0.7083333
```

Linear Kernal SVM

```
## SVM #85.4% Accurate
set.seed(143)
lin.svm <- tune.svm(MeanCount~.,data=train, kernel="linear",
                    cost=c(0.001,0.01,0.1,1,5,10,100,1000))
best.linsvm <- lin.svm$best.model
lintune.test <- predict(best.linsvm, newdata=test)
confusionMatrix(test$MeanCount,lintune.test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Hi Lo
##         Hi  7  7
##         Lo  0 34
##
##                Accuracy : 0.8542
##                  95% CI : (0.7224, 0.9393)
##     No Information Rate : 0.8542
##     P-Value [Acc > NIR] : 0.59902
##
##                   Kappa : 0.5862
##
##  Mcnemar's Test P-Value : 0.02334
##
##             Sensitivity : 1.0000
##             Specificity : 0.8293
##          Pos Pred Value : 0.5000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.1458
##          Detection Rate : 0.1458
##    Detection Prevalence : 0.2917
##       Balanced Accuracy : 0.9146
##
##        'Positive' Class : Hi
##
```

```
accuracy.meas(lintune.test, test$MeanCount)
```

```
##
## Call:
## accuracy.meas(response = lintune.test, predicted = test$MeanCount)
##
## Examples are labelled as positive when predicted is greater than 0.5
##
## precision: 0.854
## recall: 1.000
## F: 0.461
```

RBF Kernal SVM

```
#Radial Kernal 83.3% Accurate
set.seed(143)
rbf.svm <- tune.svm(MeanCount~.,data=train, kernel="radial",
                    gamma=c(0.1,0.5,1,2,3,4),
                    cost=c(0.001,0.01,0.1,1,5,10,100,1000))
best.rbfsvm <- rbf.svm$best.model
rbftune.test <- predict(best.rbfsvm, newdata=test)
confusionMatrix(test$MeanCount,rbftune.test)
```

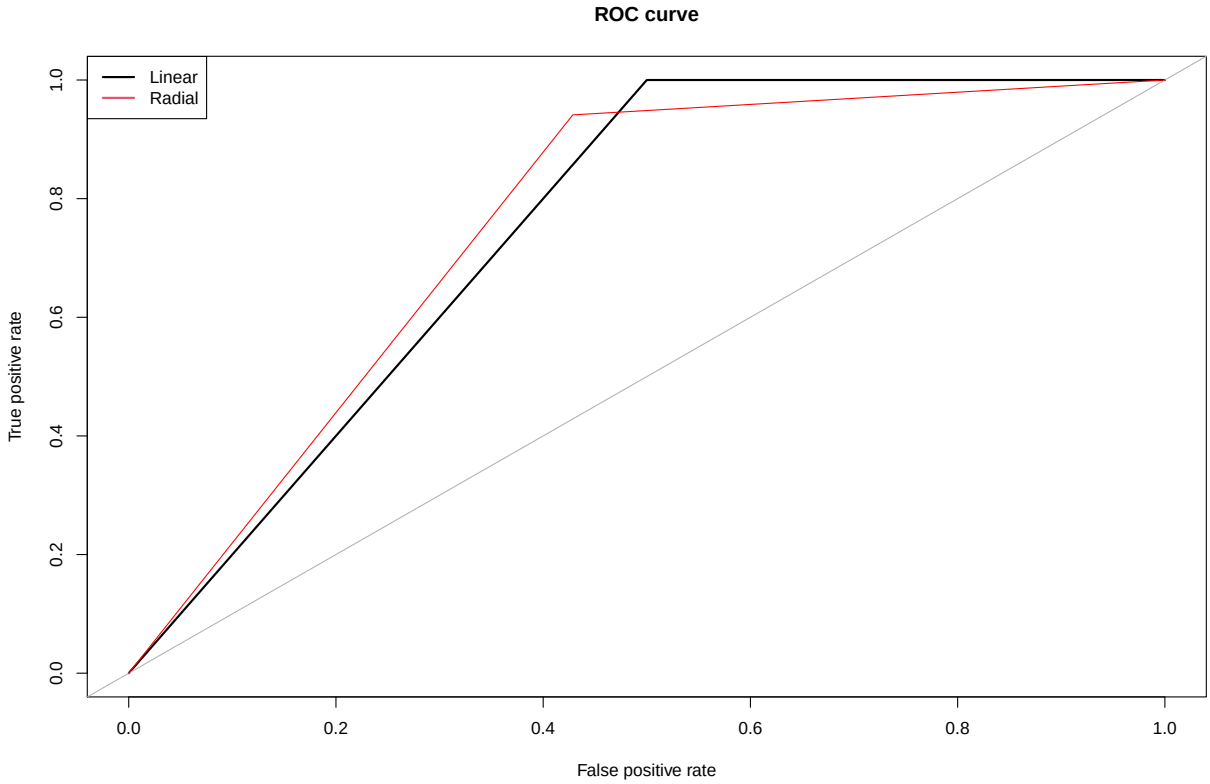
```
## Confusion Matrix and Statistics
```

```
##
##          Reference
## Prediction Hi Lo
##         Hi  8  6
##         Lo  2 32
##
##                Accuracy : 0.8333
##                  95% CI : (0.6978, 0.9252)
##     No Information Rate : 0.7917
##     P-Value [Acc > NIR] : 0.3061
##
##                   Kappa : 0.5596
##
##  Mcnemar's Test P-Value : 0.2888
##
##             Sensitivity : 0.8000
##             Specificity : 0.8421
##          Pos Pred Value : 0.5714
##          Neg Pred Value : 0.9412
##              Prevalence : 0.2083
##          Detection Rate : 0.1667
##    Detection Prevalence : 0.2917
##       Balanced Accuracy : 0.8211
##
##        'Positive' Class : Hi
##
```

```
accuracy.meas(rbftune.test, test$MeanCount)
```

```
##
## Call:
## accuracy.meas(response = rbftune.test, predicted = test$MeanCount)
##
## Examples are labelled as positive when predicted is greater than 0.5
##
## precision: 0.792
## recall: 1.000
## F: 0.442
```

```
lin.svmroc <- roc.curve(test$MeanCount, lintune.test, col="black")
rbf.svmroc <- roc.curve(test$MeanCount, rbftune.test, add.roc=TRUE, col="Red")
legend("topleft", c("Linear","Radial"), col=1:3, lty=1, lwd=2)
```




```
# Linear AUC
lin.svmroc
```

```
## Area under the curve (AUC): 0.750
```

```
# Radial AUC
rbf.svmroc
```

```
## Area under the curve (AUC): 0.756
```

Seen here, a binary classification approach greatly improved our results. At an 85.4% accuracy for the linear kernal, our output might actually be useful (Note: RBF failed the p-value). However, with a negative prediction value of 1.000 and a positive prediction value of 0.500, it might be time to address the class imbalance as we need our model to better identify positive observations. Also, as seen, our precision/recall/f-values are very satisfactory while improvements can still be made to the area under the ROC curve.

Balancing Classes

Now, I'm going to apply synthetic sampling techniques - Random Oversampling Examples (ROSE), undersampling, and oversampling. Perhaps by boosting the number of "hi" observations in the training set, we can increase our positive detection rate on the test set. Given the relatively small class imbalance, I'm not sure how effective the results will be.

```r
#Oversampling - N = 394
data_balanced_over <- ovun.sample(MeanCount ~ ., data = train, method = "over", seed = 19)$data
table(data_balanced_over$MeanCount)
```

```
##
##  Lo  Hi
## 194 200
```

```r
#Undersampling - N= 164
data_balanced_under <- ovun.sample(MeanCount ~ ., data = train, method = "under", seed = 1)$data
table(data_balanced_under$MeanCount)
```

```
##
## Lo Hi
## 81 83
```

```r
#ROSE sampling - N=150
data_balanced_rose <- ROSE(MeanCount ~ ., data = train, seed = 99, N=150)$data
table(data_balanced_rose$MeanCount)
```

```
##
## Lo Hi
## 73 77
```

Balanced Sets Applied to Both Linear and RBF SVM Kernals

Linear

```r
set.seed(132)
#Linear
linsvm.rose <- tune.svm(MeanCount~.,data=data_balanced_rose, kernel="linear",
                        cost=c(0.001,0.01,0.1,1,5,10,100,1000))
linsvm.over <- tune.svm(MeanCount~.,data=data_balanced_over, kernel="linear",
                        cost=c(0.001,0.01,0.1,1,5,10,100,1000))
linsvm.under <- tune.svm(MeanCount~.,data=data_balanced_under, kernel="linear",
                         cost=c(0.001,0.01,0.1,1,5,10,100,1000))
best.linsvm.rose <- linsvm.rose$best.model
best.linsvm.over <- linsvm.over$best.model
best.linsvm.under <- linsvm.under$best.model
lineartune.test.rose <-predict(best.linsvm.rose, newdata=test)
lineartune.test.over <- predict(best.linsvm.over, newdata=test)
lineartune.test.under <- predict(best.linsvm.under, newdata=test)
confusionMatrix(lineartune.test.rose, test$MeanCount)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Hi Lo
##         Hi 12 17
```

```
##         Lo  2 17
##
##                Accuracy : 0.6042
##                  95% CI : (0.4527, 0.7423)
##     No Information Rate : 0.7083
##     P-Value [Acc > NIR] : 0.956483
##
##                   Kappa : 0.2716
##
##  Mcnemar's Test P-Value : 0.001319
##
##             Sensitivity : 0.8571
##             Specificity : 0.5000
##          Pos Pred Value : 0.4138
##          Neg Pred Value : 0.8947
##              Prevalence : 0.2917
##          Detection Rate : 0.2500
##    Detection Prevalence : 0.6042
##       Balanced Accuracy : 0.6786
##
##        'Positive' Class : Hi
##
```

```
confusionMatrix(lineartune.test.over, test$MeanCount)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Hi Lo
##         Hi  9  7
##         Lo  5 27
##
##                Accuracy : 0.75
##                  95% CI : (0.604, 0.8636)
##     No Information Rate : 0.7083
##     P-Value [Acc > NIR] : 0.3234
##
##                   Kappa : 0.4194
##
##  Mcnemar's Test P-Value : 0.7728
##
##             Sensitivity : 0.6429
##             Specificity : 0.7941
##          Pos Pred Value : 0.5625
##          Neg Pred Value : 0.8438
##              Prevalence : 0.2917
##          Detection Rate : 0.1875
##    Detection Prevalence : 0.3333
##       Balanced Accuracy : 0.7185
##
##        'Positive' Class : Hi
##
```

```r
confusionMatrix(lineartune.test.under, test$MeanCount)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Hi Lo
##         Hi 10 13
##         Lo  4 21
##
##                Accuracy : 0.6458
##                  95% CI : (0.4946, 0.7784)
##     No Information Rate : 0.7083
##     P-Value [Acc > NIR] : 0.86601
##
##                   Kappa : 0.2792
##
##  Mcnemar's Test P-Value : 0.05235
##
##             Sensitivity : 0.7143
##             Specificity : 0.6176
##          Pos Pred Value : 0.4348
##          Neg Pred Value : 0.8400
##              Prevalence : 0.2917
##          Detection Rate : 0.2083
##    Detection Prevalence : 0.4792
##       Balanced Accuracy : 0.6660
##
##        'Positive' Class : Hi
##
```

```r
accuracy.meas(lineartune.test.rose, test$MeanCount)
```

```
##
## Call:
## accuracy.meas(response = lineartune.test.rose, predicted = test$MeanCount)
##
## Examples are labelled as positive when predicted is greater than 0.5
##
## precision: 0.604
## recall: 1.000
## F: 0.377
```

```r
accuracy.meas(lineartune.test.over, test$MeanCount)
```

```
##
## Call:
## accuracy.meas(response = lineartune.test.over, predicted = test$MeanCount)
##
## Examples are labelled as positive when predicted is greater than 0.5
##
## precision: 0.333
## recall: 1.000
## F: 0.250
```

```
accuracy.meas(lineartune.test.under, test$MeanCount)
```

```
##
## Call:
## accuracy.meas(response = lineartune.test.under, predicted = test$MeanCount)
##
## Examples are labelled as positive when predicted is greater than 0.5
##
## precision: 0.479
## recall: 1.000
## F: 0.324
```

```
rose.lin.svmroc <- roc.curve(test$MeanCount, lineartune.test.rose,
                             col="black")
over.lin.svmroc <- roc.curve(test$MeanCount, lineartune.test.over,
                             add.roc = T, col="red")
under.lin.svmroc <- roc.curve(test$MeanCount, lineartune.test.under,
                              add.roc = T, col="green")
rose.lin.svmroc
```

```
## Area under the curve (AUC): 0.679
```

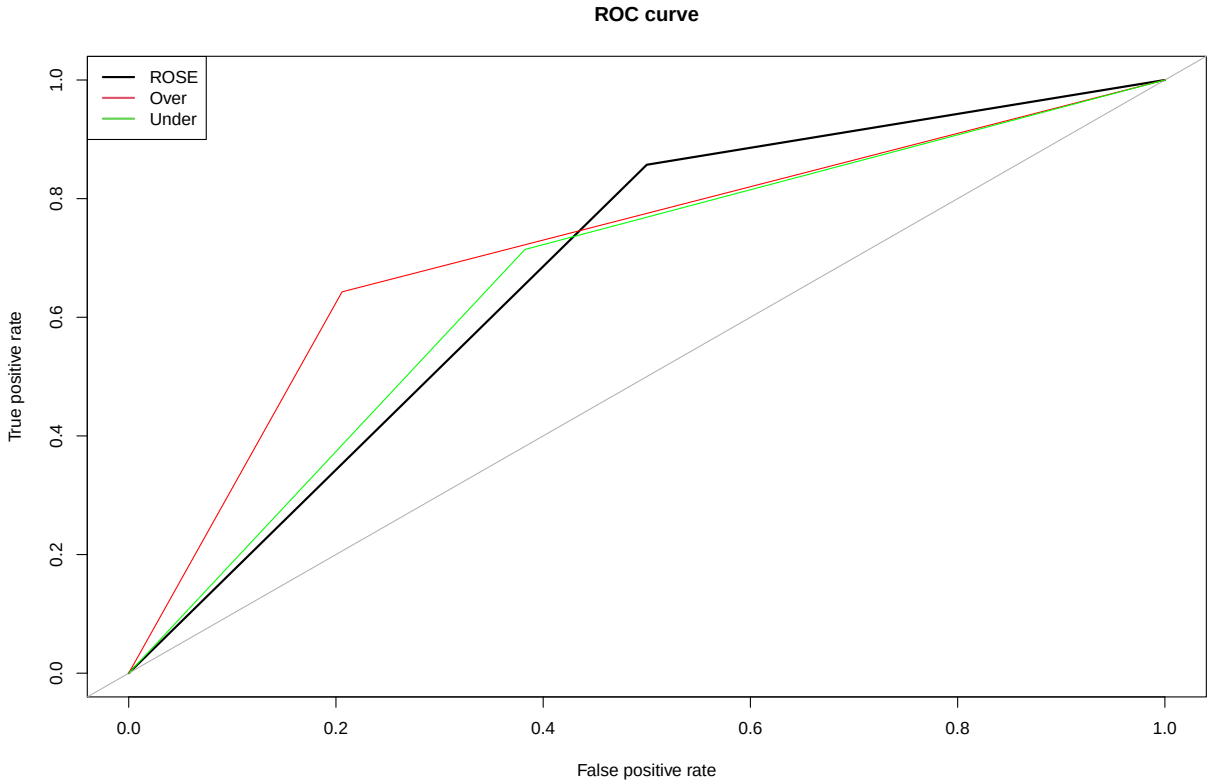
```
over.lin.svmroc
```

```
## Area under the curve (AUC): 0.718
```

```
under.lin.svmroc
```

```
## Area under the curve (AUC): 0.666
```

```
legend("topleft", c("ROSE","Over","Under"), col=1:4, lty=1, lwd=2)
```



Radial

```
#Radial Kernal
rbfsvm.rose <- tune.svm(MeanCount~.,data=data_balanced_rose, kernel="radial",
                        gamma=c(0.1,0.5,1,2,3,4),
                        cost=c(0.001,0.01,0.1,1,5,10,100))
rbfsvm.over <- tune.svm(MeanCount~.,data=data_balanced_over, kernel="radial",
                        gamma=c(0.1,0.5,1,2,3,4),
                        cost=c(0.001,0.01,0.1,1,5,10,100))
rbfsvm.under <- tune.svm(MeanCount~.,data=data_balanced_under, kernel="radial",
                         gamma=c(0.1,0.5,1,2,3,4),
                         cost=c(0.001,0.01,0.1,1,5,10,100))
best.rbfsvm.rose <- rbfsvm.rose$best.model
best.rbfsvm.over <- rbfsvm.over$best.model
best.rbfsvm.under <- rbfsvm.under$best.model
rbftune.test.rose = predict(best.rbfsvm.rose, newdata=test)
rbftune.test.over = predict(best.rbfsvm.over, newdata=test)
rbftune.test.under = predict(best.rbfsvm.under, newdata=test)
confusionMatrix(rbftune.test.rose,test$MeanCount)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Hi Lo
##         Hi 13 14
```

```
##         Lo  1 20
##
##                Accuracy : 0.6875
##                  95% CI : (0.5375, 0.8134)
##     No Information Rate : 0.7083
##     P-Value [Acc > NIR] : 0.688944
##
##                   Kappa : 0.4059
##
##  Mcnemar's Test P-Value : 0.001946
##
##             Sensitivity : 0.9286
##             Specificity : 0.5882
##          Pos Pred Value : 0.4815
##          Neg Pred Value : 0.9524
##              Prevalence : 0.2917
##          Detection Rate : 0.2708
##    Detection Prevalence : 0.5625
##       Balanced Accuracy : 0.7584
##
##        'Positive' Class : Hi
##
```

```
confusionMatrix(rbftune.test.over,test$MeanCount)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Hi Lo
##         Hi  3  0
##         Lo 11 34
##
##                Accuracy : 0.7708
##                  95% CI : (0.6269, 0.8797)
##     No Information Rate : 0.7083
##     P-Value [Acc > NIR] : 0.216188
##
##                   Kappa : 0.2787
##
##  Mcnemar's Test P-Value : 0.002569
##
##             Sensitivity : 0.2143
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.7556
##              Prevalence : 0.2917
##          Detection Rate : 0.0625
##    Detection Prevalence : 0.0625
##       Balanced Accuracy : 0.6071
##
##        'Positive' Class : Hi
##
```

```
confusionMatrix(rbftune.test.under,test$MeanCount)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Hi Lo
##         Hi 12 10
##         Lo  2 24
##
##                Accuracy : 0.75
##                  95% CI : (0.604, 0.8636)
##     No Information Rate : 0.7083
##     P-Value [Acc > NIR] : 0.32340
##
##                   Kappa : 0.482
##
##  Mcnemar's Test P-Value : 0.04331
##
##             Sensitivity : 0.8571
##             Specificity : 0.7059
##          Pos Pred Value : 0.5455
##          Neg Pred Value : 0.9231
##              Prevalence : 0.2917
##          Detection Rate : 0.2500
##    Detection Prevalence : 0.4583
##       Balanced Accuracy : 0.7815
##
##        'Positive' Class : Hi
##
```

```
accuracy.meas(rbftune.test.rose, test$MeanCount)
```

```
##
## Call:
## accuracy.meas(response = rbftune.test.rose, predicted = test$MeanCount)
##
## Examples are labelled as positive when predicted is greater than 0.5
##
## precision: 0.562
## recall: 1.000
## F: 0.360
```

```
accuracy.meas(rbftune.test.over, test$MeanCount)
```

```
##
## Call:
## accuracy.meas(response = rbftune.test.over, predicted = test$MeanCount)
##
## Examples are labelled as positive when predicted is greater than 0.5
##
## precision: 0.062
## recall: 1.000
## F: 0.059
```

```
accuracy.meas(rbftune.test.under, test$MeanCount)
```

```
##
## Call:
## accuracy.meas(response = rbftune.test.under, predicted = test$MeanCount)
##
## Examples are labelled as positive when predicted is greater than 0.5
##
## precision: 0.458
## recall: 1.000
## F: 0.314
```

```
rose.rbf.svmroc <- roc.curve(test$MeanCount, rbftune.test.rose,
                             col="black")
over.rbf.svmroc <- roc.curve(test$MeanCount, rbftune.test.over,
                             add.roc = T, col="red")
under.rbf.svmroc <- roc.curve(test$MeanCount, rbftune.test.under,
                              add.roc = T, col="green")
rose.rbf.svmroc
```

```
## Area under the curve (AUC): 0.758
```

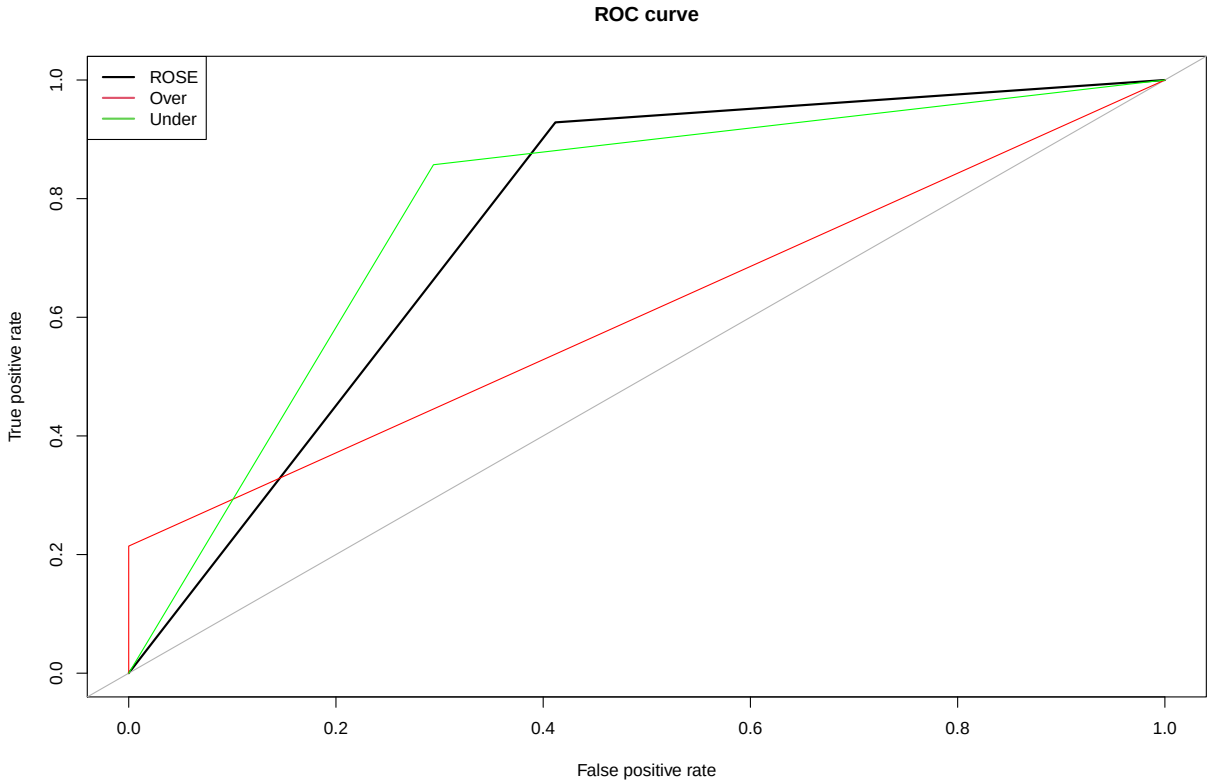
```
over.rbf.svmroc
```

```
## Area under the curve (AUC): 0.607
```

```
under.rbf.svmroc
```

```
## Area under the curve (AUC): 0.782
```

```
legend("topleft", c("ROSE","Over","Under"), col=1:4, lty=1, lwd=2)
```



When applying the balancing methods to linear and RBF SVMs, performance decreased - both in terms of accuracy and our ROC measures. Sparing any higher level tuning and risking overfitting, it appears the linear SVM using the unbalanced training set yielded the greatest results for binary classification. To push boundaries a little further, I'm going to approach this as a multi-class problem using three classes - lo, mid, and hi.

Multiple Classes Employed with SVMs

I will be binning counts greater than 100 as "Hi, 20-99 as"Mid", and 1-19 as"Lo". This approach could be very useful in flagging concerning environmental conditions for mosquito densities, while also identifying weather that is of least concern. As you might imagine, this multi-class problem produces slightly more class imbalance than the binary approach. Therefore, after applying imbalanced training sets to linear and radial SVMs, I'll apply some basic synthetic balancing methods (under and over sampling). Because we're working with 3 classes as opposed to 2, we are more limited in our balancing techniques as ROSE and SMOTE are most easily applied in binary decision making.

Data Read-In, Training/Test Partioning

```
#Machine Learning
mos<- read.csv("mosq.csv", header=T)
mos$Date<-NULL
mos$Week<-as.factor(mos$Week)
mos$Month<-as.factor(mos$Month)
mos$MeanCount<-signif(mos$MeanCount, digits=2)
mos$MeanCount <- ifelse(mos$MeanCount > 99, "Hi",
```

```
                        ifelse(mos$MeanCount >19, "Mid", "Lo"))
mos$MeanCount<-as.factor(mos$MeanCount)
preProcValues <- preProcess(mos, method = c("center", "scale"))
mos <- predict(preProcValues, mos)
set.seed(45)
trainIndex <- createDataPartition(mos$MeanCount, p = 0.85, list = FALSE, times = 1)
#Parse out training/test sets
train <- mos[ trainIndex,]
test <- mos[-trainIndex,]
#Check class distribution
prop.table(table(train$MeanCount))
```

```
##
##        Hi        Lo       Mid
## 0.2563177 0.1660650 0.5776173
```

```
prop.table(table(test$MeanCount))
```

```
##
##        Hi        Lo       Mid
## 0.2500000 0.1666667 0.5833333
```

Imbalanced Linear Kernal

```
set.seed(143)
lin.svm <- tune.svm(MeanCount~.,data=train, kernel="linear",
                    cost=c(0.001,0.01,0.1,1,5,10,100,1000))
best.linsvm <- lin.svm$best.model
lintune.test <- predict(best.linsvm, newdata=test)
confusionMatrix(test$MeanCount,lintune.test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Hi Lo Mid
##        Hi   6  0   6
##        Lo   1  1   6
##        Mid  2  1  25
##
## Overall Statistics
##
##                Accuracy : 0.6667
##                  95% CI : (0.5159, 0.796)
##     No Information Rate : 0.7708
##     P-Value [Acc > NIR] : 0.96617
##
##                   Kappa : 0.3287
##
##  Mcnemar's Test P-Value : 0.08689
##
```

```
## Statistics by Class:
##
##                      Class: Hi Class: Lo Class: Mid
## Sensitivity             0.6667   0.50000     0.6757
## Specificity             0.8462   0.84783     0.7273
## Pos Pred Value          0.5000   0.12500     0.8929
## Neg Pred Value          0.9167   0.97500     0.4000
## Prevalence              0.1875   0.04167     0.7708
## Detection Rate          0.1250   0.02083     0.5208
## Detection Prevalence    0.2500   0.16667     0.5833
## Balanced Accuracy       0.7564   0.67391     0.7015
```

Imbalanced Radial Kernal

```
set.seed(143)
rbf.svm <- tune.svm(MeanCount~.,data=train, kernel="radial",
                    gamma=c(0.1,0.5,1,2,3,4),
                    cost=c(0.001,0.01,0.1,1,5,10,100,1000))
best.rbfsvm <- rbf.svm$best.model
rbftune.test <- predict(best.rbfsvm, newdata=test)
confusionMatrix(test$MeanCount,rbftune.test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Hi Lo Mid
##        Hi   2  0  10
##        Lo   0  0   8
##        Mid  1  0  27
##
## Overall Statistics
##
##                Accuracy : 0.6042
##                  95% CI : (0.4527, 0.7423)
##     No Information Rate : 0.9375
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0952
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Hi Class: Lo Class: Mid
## Sensitivity            0.66667        NA     0.6000
## Specificity            0.77778    0.8333     0.6667
## Pos Pred Value         0.16667        NA     0.9643
## Neg Pred Value         0.97222        NA     0.1000
## Prevalence             0.06250    0.0000     0.9375
## Detection Rate         0.04167    0.0000     0.5625
## Detection Prevalence   0.25000    0.1667     0.5833
## Balanced Accuracy      0.72222        NA     0.6333
```

Multiclass Balancing

```
set.seed(234)
#Undersampling training set
threedtrain <- downSample(x = train[2:44],y = train$MeanCount)
table(threedtrain$Class)
```

```
##
##  Hi  Lo Mid
##  46  46  46
```

```
colnames(threedtrain)[44]<- "MeanCount"
#Oversampling training set
threeutrain <- upSample(x = train[2:44],y = train$MeanCount)
table(threeutrain$Class)
```

```
##
##  Hi  Lo Mid
## 160 160 160
```

```
colnames(threeutrain)[44]<- "MeanCount"
```

Balanced Linear Kernal (Over and Under Sampled)

```
set.seed(143)
#Over Sampled Training Predictions
lin.svm <- tune.svm(MeanCount~.,data=threeutrain, kernel="linear",
                    cost=c(0.001,0.01,0.1,1,5,10,100,1000))
best.linsvm <- lin.svm$best.model
lintune.test <- predict(best.linsvm, newdata=test)
confusionMatrix(test$MeanCount,lintune.test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Hi Lo Mid
##        Hi   7  0   5
##        Lo   1  1   6
##        Mid  8  6  14
##
## Overall Statistics
##
##                Accuracy : 0.4583
##                  95% CI : (0.3137, 0.6083)
##     No Information Rate : 0.5208
##     P-Value [Acc > NIR] : 0.8440
##
##                   Kappa : 0.0796
##
```

```
##  Mcnemar's Test P-Value : 0.6386
##
## Statistics by Class:
##
##                      Class: Hi Class: Lo Class: Mid
## Sensitivity             0.4375   0.14286     0.5600
## Specificity             0.8438   0.82927     0.3913
## Pos Pred Value          0.5833   0.12500     0.5000
## Neg Pred Value          0.7500   0.85000     0.4500
## Prevalence              0.3333   0.14583     0.5208
## Detection Rate          0.1458   0.02083     0.2917
## Detection Prevalence    0.2500   0.16667     0.5833
## Balanced Accuracy       0.6406   0.48606     0.4757
```

```
#Under Sampled Training Predictions
lin.svm <- tune.svm(MeanCount~.,data=threedtrain, kernel="linear",
                    cost=c(0.001,0.01,0.1,1,5,10,100,1000))
best.linsvm <- lin.svm$best.model
lintune.test <- predict(best.linsvm, newdata=test)
confusionMatrix(test$MeanCount,lintune.test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Hi Lo Mid
##        Hi   6  1   5
##        Lo   2  2   4
##        Mid  8 12   8
##
## Overall Statistics
##
##                Accuracy : 0.3333
##                  95% CI : (0.204, 0.4841)
##     No Information Rate : 0.3542
##     P-Value [Acc > NIR] : 0.6701
##
##                   Kappa : -0.0132
##
##  Mcnemar's Test P-Value : 0.1699
##
## Statistics by Class:
##
##                      Class: Hi Class: Lo Class: Mid
## Sensitivity             0.3750   0.13333     0.4706
## Specificity             0.8125   0.81818     0.3548
## Pos Pred Value          0.5000   0.25000     0.2857
## Neg Pred Value          0.7222   0.67500     0.5500
## Prevalence              0.3333   0.31250     0.3542
## Detection Rate          0.1250   0.04167     0.1667
## Detection Prevalence    0.2500   0.16667     0.5833
## Balanced Accuracy       0.5938   0.47576     0.4127
```

Balanced Radial Kernal (Over and Under Sampled)

```
set.seed(143)
#Over Sampled Training Predictions
rbf.svm <- tune.svm(MeanCount~.,data=threeutrain, kernel="radial",
                    gamma=c(0.1,0.5,1,2,3,4),
                    cost=c(0.001,0.01,0.1,1,5,10,100,1000))
best.rbfsvm <- rbf.svm$best.model
rbftune.test <- predict(best.rbfsvm, newdata=test)
confusionMatrix(test$MeanCount,rbftune.test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Hi Lo Mid
##        Hi   0  0  12
##        Lo   0  0   8
##        Mid  0  0  28
##
## Overall Statistics
##
##                Accuracy : 0.5833
##                  95% CI : (0.4321, 0.7239)
##     No Information Rate : 1
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Hi Class: Lo Class: Mid
## Sensitivity                 NA        NA     0.5833
## Specificity               0.75    0.8333         NA
## Pos Pred Value              NA        NA         NA
## Neg Pred Value              NA        NA         NA
## Prevalence                0.00    0.0000     1.0000
## Detection Rate            0.00    0.0000     0.5833
## Detection Prevalence      0.25    0.1667     0.5833
## Balanced Accuracy           NA        NA         NA
```

```
#Under Sampled Training Predictions
set.seed(143)
rbf.svm <- tune.svm(MeanCount~.,data=threedtrain, kernel="radial",
                    gamma=c(0.1,0.5,1,2,3,4),
                    cost=c(0.001,0.01,0.1,1,5,10,100,1000))
best.rbfsvm <- rbf.svm$best.model
rbftune.test <- predict(best.rbfsvm, newdata=test)
confusionMatrix(test$MeanCount,rbftune.test)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction Hi Lo Mid
##        Hi   5  1   6
##        Lo   0  3   5
##        Mid  4 11  13
##
## Overall Statistics
##
##                Accuracy : 0.4375
##                  95% CI : (0.2948, 0.5882)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.8438
##
##                   Kappa : 0.0769
##
##  Mcnemar's Test P-Value : 0.3018
##
## Statistics by Class:
##
##                      Class: Hi Class: Lo Class: Mid
## Sensitivity             0.5556    0.2000     0.5417
## Specificity             0.8205    0.8485     0.3750
## Pos Pred Value          0.4167    0.3750     0.4643
## Neg Pred Value          0.8889    0.7000     0.4500
## Prevalence              0.1875    0.3125     0.5000
## Detection Rate          0.1042    0.0625     0.2708
## Detection Prevalence    0.2500    0.1667     0.5833
## Balanced Accuracy       0.6880    0.5242     0.4583
```

Again, the unbalanced linear SVM performed the best out of all our models. However, 66.7% accuracy rate isn't very useful and our p-value violates our 0.05 threshold of significance. In addition while easily identifying the 'mid' classes, this model performed extremely poorly on categorizing 'lo' mosquito counts.

Discussion

I did explore other learning algorithms, such decision trees and random forests, and I attempted to employ ensemble methods for boosting the results of my SVMs. However, none of these processes yielded results that exceeded the basic linear SVM and our unaltered training set. By all accuracy and confidence measures, the linear SVM would be the method I'd choose going forward. I know there exists a plethora of other machine learning and AI techniques that might more gracefully handle my dataset, but it became too computationally exhausting to test so many other methods and those approaches sometimes fell outside of my current skill level.

To improve my dataset and perhaps boost my results, I could consider removing all stream data because it proved to be an insignificant contributor to mosquito density. Also, because scarcity in the stream gage data caused me to narrow my entire project to 2017-2020, I could go back and add the years 2015 and 2016. Having 5 years of data (2015-2020) would allow me to create larger training and test sets which would correct learning deficiencies in my completed models.

Often a problem in data mining, the DEPs mosquito dataset is maintained in coordination with monitoring programs throughout counties in the commonwealth of Pennsylvania. The data wasn't necessarily collected and compiled with the implied purpose of being used in machine algorithms. It would be prohibitively costly and inefficient to monitor environmental conditions every day at every site for five years,

so I largely generalized all of my weather data off of the mentioned airport stations. In the past, funded research in Canada and metro-Chicago has built in these kinds of scientific qualitative controls with much better results.

Finally, in the future and with many more years of data, I think it'd be interesting to include daily Positive West Nile cases as a target variable. I know the daily number of mosquitos caught and the time of year are highly correlated to incidences of West Nile Virus, and I feel my machine learning techniques would be especially useful in providing early warning of positive WNV cases prior to the test results returning from a lab.

Works Cited

Andreadis, T., 2012. The contribution of Culex pipiens complex mosquitos to transmission and persistance of West Nile virus in North America. Journal of American Mosquito Control Association, 28(4s), pp. 137-151.

Control., E.C.f.D.P.a., 2014. Annual epidemiological report 2014-emerging and vector-borne diseases, Stockholm: ECDC.

Crans, W., 2004. A classification system for mosquito life cycles: life cycle types for mosquitoes of the northeastern United States. Journal of Vector Ecology, Volume 29, pp. 1-10.

Harrington, L. & Poulson, R., 2008. Considerations for accurate identification of adult Culex restuans (Diptera: Culicidae) in field studies. Journal of Medical Entomology, Volume 45, pp. 1-8.

Jones, C., Lounibos, L., Marra, P. & Kilpatrick, A., 2012. Rainfall influences survival of Culex pipiens (Diptera: Culicidae) in a residential neighborhood in the mid-Atlantic United States. Journal of Medical Entomology, 49(3), pp. 467-473.

Karki, S. et al., 2016. Effect of Trapping Methods, Weather, and Landscape on Estimates of the Culex Vector Mosquito Abundance. Environmental Health Insights, 20 May, Volume 10, pp. 93-103.

Nash, D. et al., 2001. The outbreak of WNV infection in the New York City area in 1999.. New England Journal of Medicine, Volume 344, pp. 807-814.

Reisen, W. K. et al., 2006. Role of corvids in epidemiology of West Nile Virus in southern California. Journal of Medical Entomology, Volume 43, pp. 356-367.