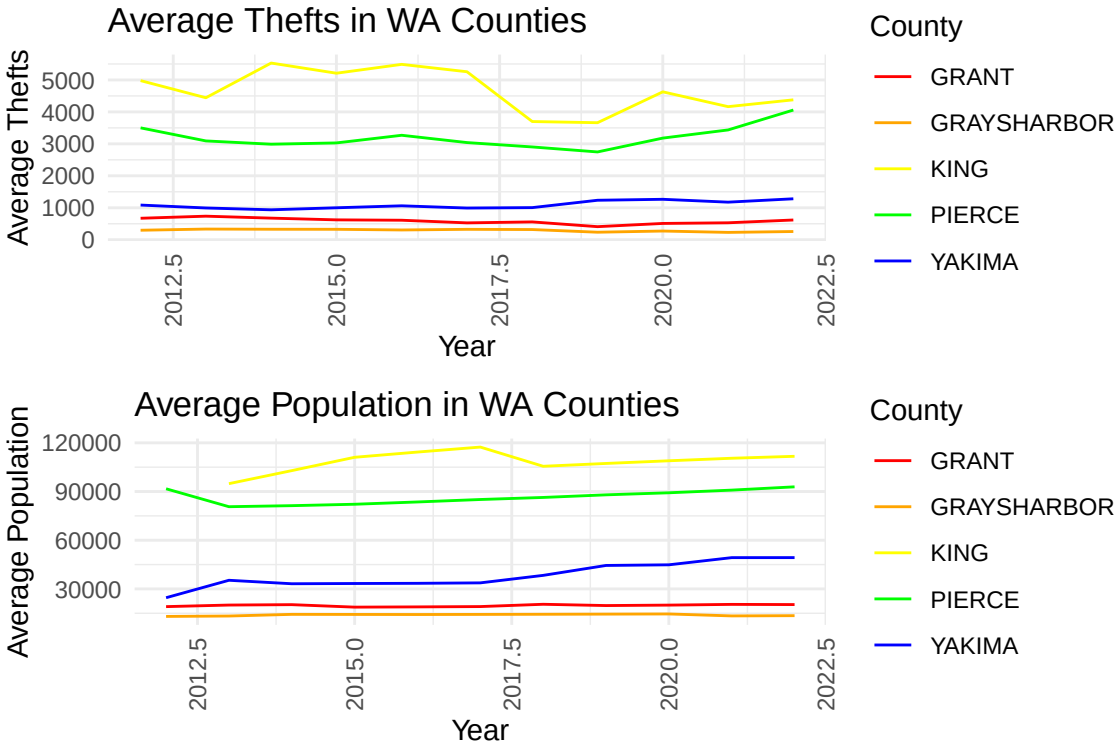
Using Classification to Inform Public Safety

Which Counties in WA are Most Likely to Contain Low Crime?

Suppose you are moving to, or living in WA, and you want to ensure that the location you live is safe. Naturally, you'd want to ensure this means it has low crime rates. Perhaps you're specific question is, which counties in WA are predicted to have the highest crime? Knowing this can help you navigate which counties you don't want to live in. We can answer this question using a predictive modeling technique called Support Vector Machine Learning (SVM). Before diving into this concept, lets first upload the data and get familiar with it! For this data, a given observation depicts the counts of different crime-types in each county of WA in a given year, among other variables such as population, city police department, percent of people living in that county, etc.

```
#install.packages("readxl") #install the required library to upload the data
library(readxl) #upload the library
#identify the path on your Excel data
directory="C:/Users/eliza/OneDrive/Documents/WADEPS/Weekly notes an ideas/Classification_SVM_Example_04_
data=read_excel(directory) #upload your data set
```

Let's visualize a line plot depicting counts of theft, kidnapping, and human trafficking in Grant, Grays Harbor, Pierce, King, and Yakima counties.



Cleaning Our Data

The first step to any data analysis is to clean our data. This is just a fancy way of saying that we need to make it suitable to plug into our SVM model. We specifically need to remove any observations with blank entries. These correspond to rows with "NA" in them. Let's check if there are any such rows.

```
NA_row_numbers=!complete.cases(data) #identify row numbers in the data with
  #empty entries
NA_rows=data[NA_row_numbers,] #identify all rows in the data with empty entries
dim(NA_rows)[1] #count the number of rows with empty entries
```

```
## [1] 27
```

As we can see, there are 27 rows with empty entries. In case you need convincing, let's visualize one of these rows.

```
NA_rows[1,] #output the first row in data with empty entry(ies)
```

```
## # A tibble: 1 x 34
##   IndexYear county Location     Population Total  Rate PrsnTotal PrsnRate Murder
##   <chr>     <chr>  <chr>             <dbl> <dbl> <dbl>     <dbl>    <dbl>  <dbl>
## 1 2012      CLARK  WSU Vancouv~         NA     9    NA         1       NA      0
## # i 25 more variables: Manslaughter <dbl>, Forcible_Sex <dbl>, Assault <dbl>,
## #   Non_Forcible_Sex <dbl>, Kidnapping_Abduction <dbl>,
## #   Human_Trafficking <dbl>, Viol_Of_No_Contact <dbl>, PrprtyTotal <dbl>,
## #   PrprtyRate <dbl>, Arson <dbl>, Bribery <dbl>, Burglary <dbl>,
## #   Counterfeiting_Forgery <dbl>, Destruction_of_Property <dbl>,
## #   Extortion_Blackmail <dbl>, Robbery <dbl>, Theft <dbl>, SctyTotal <dbl>,
## #   SctyRate <dbl>, Drug_Violations <dbl>, Gambling_Violations <dbl>, ...
```

The first observation from Vancouver WA contains its first empty entry "NA" in the population column. We want to remove all such rows from our data. Let's do this, and then visualize a few of the remaining observations.

```
data_removed_NAs=data[!NA_row_numbers,] #remove NA rows from the data
data_removed_NAs #output the remaining data
```

```
## # A tibble: 2,880 x 34
##    IndexYear county Location    Population Total  Rate PrsnTotal PrsnRate Murder
##    <chr>     <chr>  <chr>            <dbl> <dbl> <dbl>     <dbl>    <dbl>  <dbl>
##  1 2012      ADAMS  Adams Coun~       9860   620  62.9       145     14.7      0
##  2 2013      ADAMS  Adams Coun~       9935   693  69.8       130     13.1      2
##  3 2014      ADAMS  Adams Coun~      10025   688  68.6       165     16.5      1
##  4 2015      ADAMS  Adams Coun~       9960   685  68.8       139     14.0      0
##  5 2016      ADAMS  Adams Coun~       9975   571  57.2       166     16.6      1
##  6 2017      ADAMS  Adams Coun~      10035   498  49.6       133     13.3      1
##  7 2018      ADAMS  Adams Coun~      10090   456  45.2       142     14.1      0
##  8 2019      ADAMS  Adams Coun~      10145   389  38.3       118     11.6      0
##  9 2020      ADAMS  Adams Coun~      10275   438  42.6       129     12.6      0
## 10 2021      ADAMS  Adams Coun~      10410   677  65.0       181     17.4      2
## # i 2,870 more rows
## # i 25 more variables: Manslaughter <dbl>, Forcible_Sex <dbl>, Assault <dbl>,
## #   Non_Forcible_Sex <dbl>, Kidnapping_Abduction <dbl>,
## #   Human_Trafficking <dbl>, Viol_Of_No_Contact <dbl>, PrprtyTotal <dbl>,
## #   PrprtyRate <dbl>, Arson <dbl>, Bribery <dbl>, Burglary <dbl>,
## #   Counterfeiting_Forgery <dbl>, Destruction_of_Property <dbl>,
## #   Extortion_Blackmail <dbl>, Robbery <dbl>, Theft <dbl>, SctyTotal <dbl>, ...
```

We now have 2880 observations out of the original 2907 with non empty entries. Now that we have cleaned our data, lets familiarize ourselves with SVM.

What is SVM Learning?

SVM learning is a method used to predict the whether or not an event occurs, given a set of data. Think of it as this: you plug in a set of data to a predictive function you are building, your function learns the patterns of yes/no events of a given outcome variable of interest, and then uses what it learned to predict whether or not this outcome occurs on future unseen data. We provide a brief summary of the steps involved in this process, but we go into them in much more detail later. Here's the big picture- We first build predictive models with all variables in our data , and then reduce the number of predictors in those models. We then use our results from the reduction process to assign a rank to each input variable that determines how much influence it has over the outcome variable of interest. We then select those inputs for our predictive functions that have the largest rank (greatest influence on the indicated outcome variable).

Like any function, we have a set inputs we plug in, and an output that is produced. Our inputs in this case are continuous (can be integers or decimals) or binary (0-1 "no-yes") variables, and our output is a binary variable. Numerically, this means none of our input variables can have 3 or more discrete, whole-numbers, as classes of responses. For instance, if one input variable "Race" has 4 categories "White", "Black", "Asian", or "other", then we would need to convert it into four binary input variables "White", "Black", "Asian", and "other", where a 1 indicates a given race and a 0 indicates another race. An example of a binary outcome variable is "WhiteSuspect" with a response of 1 if the subject is white and 0 if not.

Exploring Crime in WA

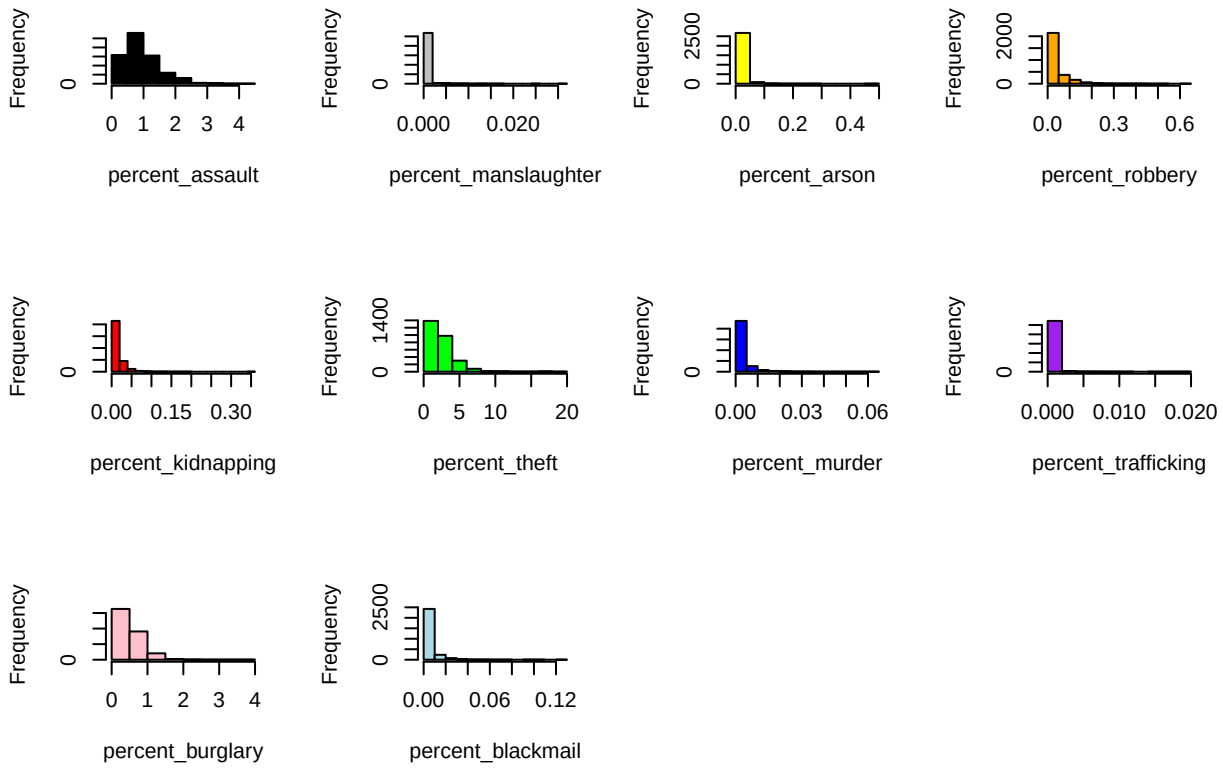
Because our research question is related to crime, lets first visualize all crime-related variables in our data set.

```
colnames(data_removed_NAs) #output the variable names in our data
```

```
##  [1] "IndexYear"                "county"
##  [3] "Location"                 "Population"
##  [5] "Total"                    "Rate"
##  [7] "PrsnTotal"                "PrsnRate"
##  [9] "Murder"                   "Manslaughter"
## [11] "Forcible_Sex"             "Assault"
## [13] "Non_Forcible_Sex"         "Kidnapping_Abduction"
## [15] "Human_Trafficking"        "Viol_Of_No_Contact"
## [17] "PrprtyTotal"              "PrprtyRate"
## [19] "Arson"                    "Bribery"
## [21] "Burglary"                 "Counterfeiting_Forgery"
## [23] "Destruction_of_Property" "Extortion_Blackmail"
## [25] "Robbery"                  "Theft"
## [27] "SctyTotal"                "SctyRate"
## [29] "Drug_Violations"          "Gambling_Violations"
## [31] "Pornography"              "Prostitution"
## [33] "Weapon_Law_Violation"     "Animal_Cruelty"
```

Suppose we are interested in whether or not a given observation reveals that over 3% of the total population contained a particular crime of interest in a given year in a given county. Our binary outcome variable can then be defined using the ratio of the number of crimes of that type to the number of people living in that county in a given year (i.e. the crime variable divided by the "Population" variable for a given observation) in our data set. Let's define this outcome variable as "percent_crime", and observe its distribution for each crime type.

```r
#compute the percent of each crime-type committed in a given year for all
#observations by dividing the counts of that crime by the total
#population size and multiplying by 100
percent_assault=(data_removed_NAs$Assault/data_removed_NAs$Population)*100
percent_manslaughter=(data_removed_NAs$Manslaughter/data_removed_NAs$Population)*100
percent_arson=(data_removed_NAs$Arson/data_removed_NAs$Population)*100
percent_robbery=(data_removed_NAs$Robbery/data_removed_NAs$Population)*100
percent_kidnapping=(data_removed_NAs$Kidnapping_Abduction/data_removed_NAs$Population)*100
percent_theft=(data_removed_NAs$Theft/data_removed_NAs$Population)*100
percent_murder=(data_removed_NAs$Murder/data_removed_NAs$Population)*100
percent_trafficking=(data_removed_NAs$Human_Trafficking/data_removed_NAs$Population)*100
percent_burglary=(data_removed_NAs$Burglary/data_removed_NAs$Population)*100
percent_blackmail=(data_removed_NAs$Extortion_Blackmail/data_removed_NAs$Population)*100

#plot a histogram to visualize the frequencies of the crime-type rates
par(mfrow=c(3,4))
hist(percent_assault,col='black',main='')
hist(percent_manslaughter,col='gray',main='')
hist(percent_arson,col='yellow',main='')
hist(percent_robbery,col='orange',main='')
hist(percent_kidnapping,col='red',main='')
hist(percent_theft,col='green',main='')
hist(percent_murder,col='blue',main='')
hist(percent_trafficking,col='purple',main='')
hist(percent_burglary,col='pink',main='')
hist(percent_blackmail,col='lightblue',main='')
```



It appears that the most-occurring crimes in WA are assault, theft, and burglary. Theft occurs the most according to our histograms, so lets select this crime as our outcome variable of interest for our SVM models. Specifically, we define our binary outcome variable as "large_theft_rate", where its response is 1 if at least %3 of the population committed theft in a given year in a given county, and 0 otherwise.

Defining Large Theft Rates

First, notice that some of our observations have "Population"=0. We of course are only interested in studying percent theft in counties with population greater than 0. These value are most likely miss-typed entries. So let's remove these from our data and check there are no observations remaining afterwards with 0 population.

```
#compute all row numbers of observations with 0 population
population_zero_row_numbers=which(data_removed_NAs$Population==0)
#remove these rows from the data
data_removed_NAs_new=data_removed_NAs[-population_zero_row_numbers,]
#check if there are remaining rows with 0 population by counting how many
#there are (should be zero)
length(which(data_removed_NAs_new$Population==0))
```

```
## [1] 0
```

As shown, we have no observations with 0 population. Now, we must re-define our percent theft variable with this updated data set.

```
#re-compute the percent of thefts committed each year with 0-populations removed
percent_theft=(data_removed_NAs_new$Theft/data_removed_NAs_new$Population)*100
```

Now we can define our outcome variable "large_theft_rate".

```
#initialize the list to store binary 0's or 1's
large_theft_rate=c()

#for each observation in our data
for (observation in 1:dim(data_removed_NAs_new)[1]){
  #if there are greater than 3% of thefts committed in that year in that county
  if (percent_theft[observation]>=3){
    #store a 1 in our outcome variable
    large_theft_rate[observation]=1}
  #otherwise, if there are less than 3% of thefts committed, store a 0
  else{large_theft_rate[observation]=0}
}
```

Let's now observe the counts of observations depicting at least 3% of the population committing theft and those that don't. We can do this by counting the total number of 1's and 0's in our outcome variable.

```
#count the total number of observations with less than 3% theft committed
length(which(large_theft_rate==0))
```

```
## [1] 2013
```

```
#count the total number of observations with at least 3% theft committed
length(which(large_theft_rate==1))
```

```
## [1] 762
```

Observe that we have 2013 observations for which less than 3% of the population committed theft, and 762 for which at least 3% of the population committed theft in a given county of WA. It is important to check that we have a decently large amount of both classes of responses in our outcome variable, so that when we

perform SVM learning, our model has enough of each outcome-type to learn patterns of responses on, as mentioned in "What is SVM Learning?" above.

Now that we have defined our binary outcome variable, we can evaluate the influence of our input variables in the data on large theft rates in each county. To do this, we can first break up our data set into subsets based on which county they come from. This information comes from the "county" column.

Exploring WA Counties Represented in Data

Lets observe the different counties represented in our data.

```
#output the unique counties represented in our data
unique(data_removed_NAs_new$county)
```

```
##  [1] "ADAMS"        "ASOTIN"       "BENTON"       "CHELAN"       "CLALLAM"
##  [6] "CLARK"        "COLUMBIA"     "COWLITZ"      "DOUGLAS"      "FERRY"
## [11] "FRANKLIN"     "GARFIELD"     "GRANT"        "GRAYS HARBOR" "ISLAND"
## [16] "JEFFERSON"    "KING"         "KITSAP"       "KITTITAS"     "KLICKITAT"
## [21] "LEWIS"        "LINCOLN"      "MASON"        "OKANOGAN"     "PACIFIC"
## [26] "PEND OREILLE" "PIERCE"       "SAN JUAN"     "SKAGIT"       "SKAMANIA"
## [31] "SNOHOMISH"    "SPOKANE"      "STATEWIDE"    "STEVENS"      "THURSTON"
## [36] "WAHKIAKUM"    "WALLA WALLA"  "WHATCOM"      "WHITMAN"      "YAKIMA"
```

As previously discussed, we want to compare theft rates in counties with sufficiently large enough observations. Otherwise, our SVM functions won't have enough data to learn the patterns of yes/no responses on. For this reason, we choose to compare counties with at least 100 observations each: Grant, Grays Harbor, King, Pierce, and Yakima county. We obtain these counts using the code below.

```
#split data observations by county
counties=split(data_removed_NAs_new,data_removed_NAs_new$county)
#obtain number of observations from each county
frequencies=sapply(counties,function(counties) nrow(counties))
frequencies #output these counts
```

```
##        ADAMS       ASOTIN       BENTON       CHELAN      CLALLAM        CLARK
##           44           44           65           32           55           88
##     COLUMBIA      COWLITZ      DOUGLAS        FERRY     FRANKLIN     GARFIELD
##           22           77           33           26           44           22
##        GRANT GRAYS HARBOR       ISLAND    JEFFERSON         KING       KITSAP
##          106          110           41           33          340           66
##     KITTITAS    KLICKITAT        LEWIS      LINCOLN        MASON     OKANOGAN
##           51           54           98           44           33           92
##      PACIFIC PEND OREILLE       PIERCE     SAN JUAN       SKAGIT     SKAMANIA
##           66           26          217           22           66           22
##    SNOHOMISH      SPOKANE    STATEWIDE      STEVENS     THURSTON    WAHKIAKUM
##          188           65           11           58           70           22
##  WALLA WALLA      WHATCOM      WHITMAN       YAKIMA
##           43           86           57          136
```

Building Our Predictive SVM Model

Now that we have our response variable (large_theft_rate) defined, and our five county subsets identified, lets re-define our data by adding this response variable to it.

```
#add our binary outcome variable large_theft_rate to our data
data_new=cbind(data_removed_NAs_new,large_theft_rate)
```

Now lets formally define subsets of data for the five counties we are comparing the presence of high theft rates in.

```
#split our data with our outcome variable included into subsets based on county
data_new_split=split(data_new,data_new$county)
#define observations from Grant county WA
Grant_obs=data_new_split$`GRANT`
#define observations from Grays Harbor county WA
GraysHarbor_obs=data_new_split$`GRAYS HARBOR`
#define observations from Pierce county WA
Pierce_obs=data_new_split$`PIERCE`
#define observations from King county WA
King_obs=data_new_split$`KING`
#define observations from Yakima county WA
Yakima_obs=data_new_split$`YAKIMA`
```

Train-Test Split: Building data for our models to learn from and predict on

In order for our models to learn patterns of yes/no responses to large_theft_rate, we must split up each of these data sets into training and testing sets. The training set is for our model to learn the patterns, and the test set is for our model to make predictions of future yes/no responses (i.e. if there will be at least 3% of theft committed in a given county in a given year). We choose to split our data set into 75% of samples for training our models, and the rest for testing our models. We illustrate an example for our Grant county data set, and repeat for the remaining four sets.

```
#GRANT COUNTY MODEL: Determine total observations in Grant county
nrow(Grant_obs)
```

```
## [1] 106
```

We have 106 samples in our Grant county data set. Now lets assign 75% of those samples as our training set and the remaining samples as our testing set.

```
#GRANT COUNTY TRAIN-TEST SPLIT

#construct training set (75% of samples)
num_training_rows=round(0.75*nrow(Grant_obs)) #determine number of training rows
#randomly sample rows from Grant data for training
training_row_numbers=sample(1:num_training_rows, num_training_rows, replace = FALSE)
#construct training set
training_Grant=Grant_obs[training_row_numbers,]

#construct testing set (remaining 25% of samples)
testing_row_numbers=setdiff(1:nrow(Grant_obs),training_row_numbers)
testing_Grant=Grant_obs[testing_row_numbers,]
```

We now repeat for the remaining four counties in our analysis.

Let's check the number of observations in our training sets.

```
#compute number of observations (rows) in each training set
nrow(training_GraysHarbor);nrow(training_Pierce);nrow(training_King); +
  nrow(training_Grant); nrow(training_Yakima)
```

```
## [1] 82
```

```
## [1] 163
```

```
## [1] 255
```

```
## [1] 80
```

```
## [1] 102
```

We need to lastly remove the "county" and "location" columns from our training and testing data, as text-variables are not usable inputs for SVM, as described in "What is SVM Learning?" above. This step is called pre-processing.

```
#Remove "county" and "Location" variables from each set of data

#Grant
training_Grant=training_Grant[,-which(names(training_Grant) %in%
                                        c("county","Location"))]
testing_Grant=testing_Grant[,-which(names(testing_Grant) %in%
                                      c("county","Location"))]

#Grays Harbor
training_GraysHarbor=training_GraysHarbor[,-which(names(training_GraysHarbor)
                                                    %in% c("county","Location"))]
testing_GraysHarbor=testing_GraysHarbor[,-which(names(testing_GraysHarbor)
                                                  %in% c("county","Location"))]

#Pierce
training_Pierce=training_Pierce[,-which(names(training_Pierce)
                                        %in% c("county","Location"))]
testing_Pierce=testing_Pierce[,-which(names(testing_Pierce)
                                      %in% c("county","Location"))]

#King
training_King=training_King[,-which(names(training_King)
                                    %in% c("county","Location"))]
testing_King=testing_King[,-which(names(testing_King)
                                  %in% c("county","Location"))]

#Yakima
training_Yakima=training_Yakima[,-which(names(training_Yakima) %in%
                                          c("county","Location"))]
testing_Yakima=testing_Yakima[,-which(names(testing_Yakima) %in%
                                        c("county","Location"))]
```

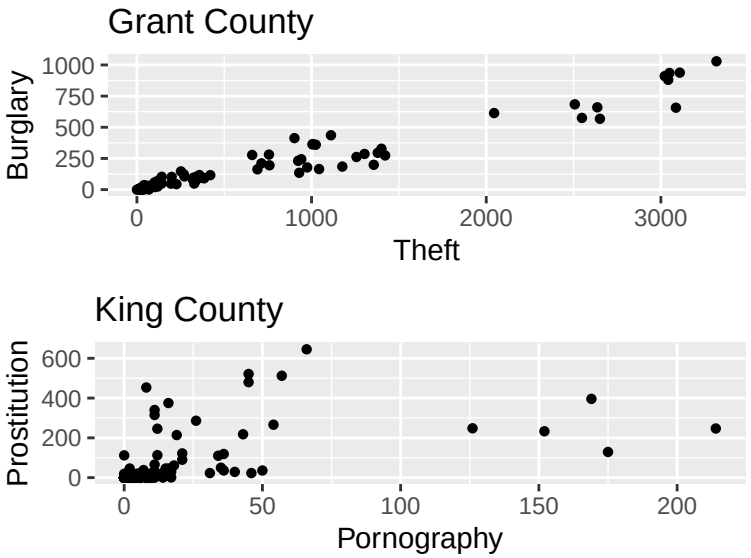
Model Parameters in SVM: What types of functions can we build?

One main benefit of using SVM, as opposed to building solely linear models used in logistic regression, is that this approach allows us to build models that define non-linear relationships between input variables and the presence of large theft rates in WA counties. Non-linear models may perform better than linear ones, which makes using SVM beneficial. That being said, lets go over the different types of functions that we can build between input variables (predictors) and outcome variables (response). We first note that SVM works by building a separating hyperplane between classes of "yes" and "no" responses (i.e. 1 and 0 values) in data during training, and predicting whether or not future observations have responses on the "yes" or "no" side of this plane during testing.

Linear: Linear models are exactly as they sound. They represent models that are able to separate both

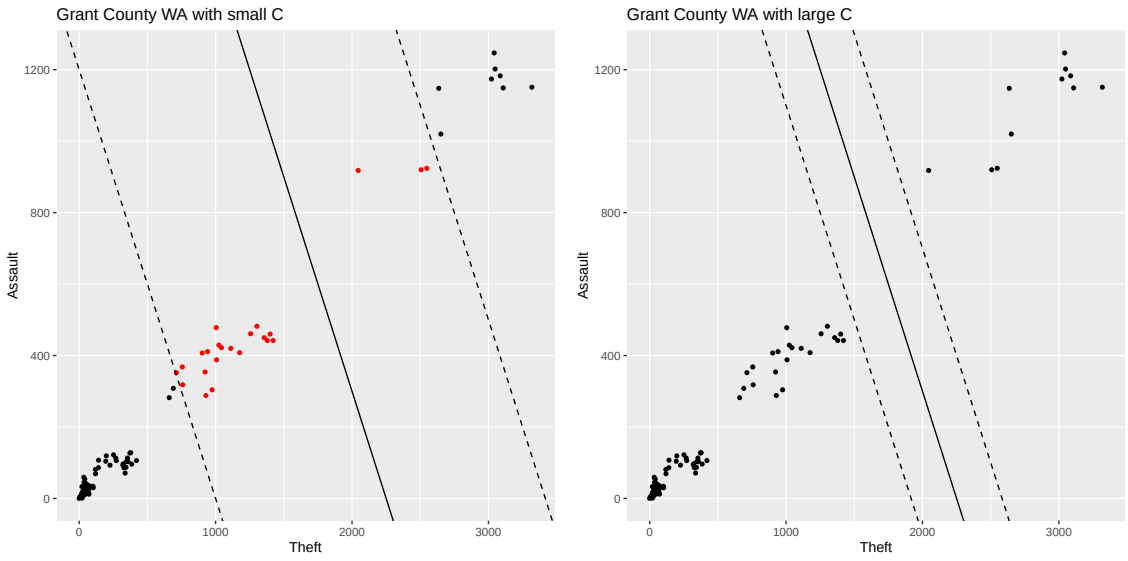
classes of responses using a line in 2D or hyperplane in dimensions greater than 3D. See an example of data that is linearly separable below.

Non-Linear models: These are again just as they sound. These models are those that are not able to linearly separate both classes of responses using a line or hyperplane. These models work by mapping the data as it is into a high-enough dimension so that its responses are linearly separable. Such models include Sigmoid, Radial Basis (rbf), and Polynomial (poly). See an example of non-linearly separable data.



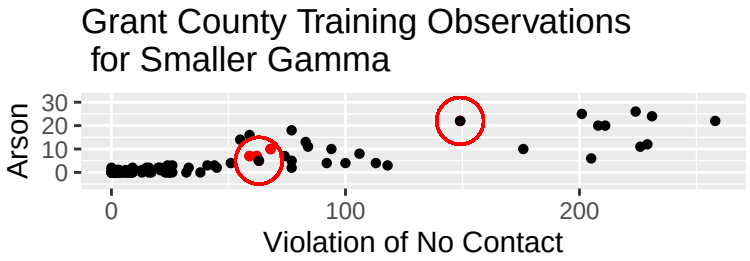
Model Parameters:

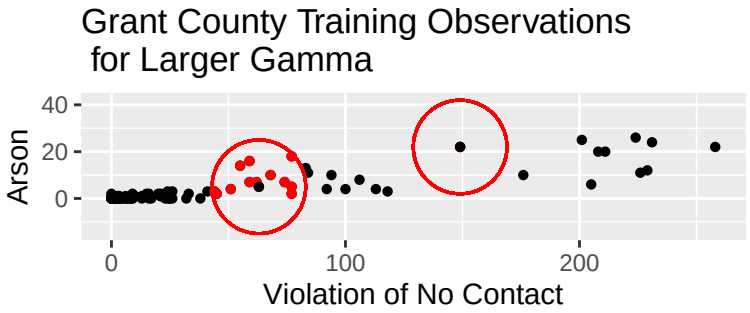
C: SVM builds a separating hyper-plane with the aim of maximizing the distance between both classes of responses, while minimizing the penalty for errors/mis-classification. C>0 is a value that balances these two goals. Small values of C increase the distance between both classes (i.e. the penalty for error) but increase the mis-classification, while large values of C decrease the mis-classification but decrease the distance (i.e. the penalty for error) between both classes. We show an example with our Grant county data below.



Gamma: If we were to plot our all observations in our Grant county training data, we would obtain a point cloud of 106, 35-dimensional data points, since this training set has 106 observations each with 35 variables. Each of these observations is called a support vector, and has some degree of influence over the presence

of large theft rates in other vectors around it within the point cloud. Gamma is a parameter that controls this degree of influence. Think of it as a radius - small values of gamma mean that for a given Grant county training observation, it has influence over those observations within a small radius of it in the point cloud. Hence it only has influence over vectors that are "close" to it. Oppositely, large values of gamma mean that a given observation has influence over observations within a large radius of them in the point cloud, and hence has influence over vectors that are "far" from it. Lets visualize what this could look like for our Grant county training data for only 2 variables. We show two instances of small gamma (radius of influence) and large gamma, where support vectors are in black and their resulting influenced vectors are in red.





Degree: Degree is a parameter reserved for the polynomial SVM model, were degree 2 indicates a quadratic relationship between the input variables and the presence of large theft rates, degree 3 indicates a cubic relationship, and so on.

As of now, there is no proven method to know which are the optimal values of these models and parameters to choose. Because of this, we employ a commonly used technique called a grid search. That is, we try various combinations of models and model parameters, and train and test our SVM model, and choose the combination of model parameters that predicts the presence of large theft rates in a given county the best. These models will contain all 35 input variables in for each of our five county models Grant, Grays Harbor,

Pierce, King, and Yakima.

Grid Search: Which Model Predicts Large Theft Rates the Best Given all Variables?

Now that our training and testing sets have been pre-processed, it is time for the fun part! Building our models. The first step is to identify the models that are the best at predicting the presence of large theft rates in each county with all variables in our data. To implement our grid search, for each combination of model parameters, we train the resulting model on the training data for each county (so it learns patterns of responses), and then test it on the testing data (so that it makes predictions of large theft rates on its responses). We then store this model's predictions and compare them to the true responses in the training data. We again show an in-depth example for Grant county, and then repeat for the remaining four counties.

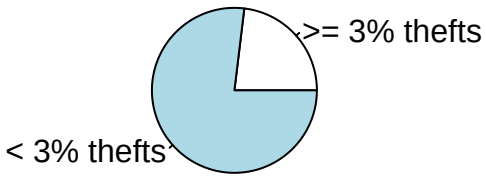
Let's first practice fitting our SVM model so that it learns the presence of large theft rates in all training observations, and then using our model to make predictions of the presence of large theft rates in all testing observations of King county. We then determine how accurate this model's predictions are using matthews correlation coefficient. This is a metric that incorporates true and false positive and negative predictions of the model to determine its accuracy. It returns a value between -1 and +1, where 0 implies the model is randomly guessing the presence of large theft rates, while +1 (-1) indicates perfect (opposite) predictions.

```
#install.packages('svm')
#install.packages("e1071")
#install.packages("mltools")

library(mltools)
library(e1071)

#define training data Grant
x=training_Grant[, -which(names(training_Grant)=="large_theft_rate"),
                 drop = FALSE]
#define testing data King
y=as.factor(training_Grant$large_theft_rate)

#combine train and test data into data frame
dat=data.frame(x,y)

#train a linear SVM model
modelfit = svm(y ~ ., data = dat, kernel = 'linear', cost = 0.01, gamma=0.13)

#test the linear SVM model
predictions=predict(modelfit, newdata=testing_Grant)

#compute predictive accuracy using MCC
score=mcc(as.numeric(predictions),testing_Grant$large_theft_rate)

score
```

```
## [1] -0.2635231
```

```
#output pie chart of model predictions with MCC score

#percent predictions that at least 3% of the Grant population will commit theft
percent.positive=round((length(predictions[predictions==1])/length(predictions))
                       *100,2)
```

```
#percent predictions that less than 3% of the Grant population will commit theft
percent.negative=round((length(predictions[predictions==0])/length(predictions))
                       *100,2)

#plot pie chart
slices=c(percent.positive,percent.negative)
labels=c(">= 3% thefts","< 3% thefts") #opposite since MCC<0
pie(slices, labels = labels, main = "")
```



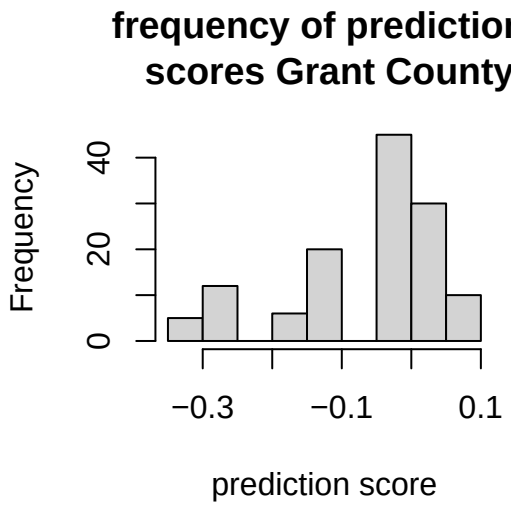
Observe that our linear SVM model is 35 percent accurate, and predicts that Grant county will contain a majority of large theft rates (at least 3% of the population). Hence our model is roughly 30% accurate and predicts a majority of low theft rates (less than 3%) in Grant county, WA.

When conducting a grid search, we just repeat this process of training and testing our models and saving performance scores for a variety of model parameters, and choose the combination that is the best at predicting the presence of large theft rates in Grant county.

```
#GRID SEARCH: Grant County
Models=c('linear','radial','sigmoid','polynomial') #optional models
Gamma=c(0.01,0.1,100,1000) #optional radii of influence
C=c(0.01,0.1,100,1000) #optional balances between max distance and
Degree=c(2,3) #optional degrees

#remove variables in Grant data with only one class of responses
training_Grant=training_Grant[,-which(names(training_Grant)==
                                      c("Human_Trafficking",
                                        "Gambling_Violations"))]
testing_Grant=testing_Grant[,-which(names(testing_Grant)==
                                      c("Human_Trafficking",
                                        "Gambling_Violations"))]

#initialize list to store models
Modellist=matrix(nrow=0,ncol=4)
#initialize list to store performance scores
MCClist=numeric()
#initialize list to store model predictions (% yes, % no)
```

```
perc.preds=matrix(nrow=0,ncol=2)


for (i in 1:length(Models)){ #for every model type
  Mod=Models[i]
  for (j in 1:length(Gamma)){ #for every gamma value
    Gam=Gamma[j]
    for (k in 1:length(C)){ #for every C value
      c=C[k]
      for (l in 1:length(Degree)){ #for every degree
        if (Mod=="polynomial"){
          Deg=Degree[l]
        }

        #define the Grant county training data
        x=training_Grant[, -which(names(training_Grant)=="large_theft_rate"),
                                                          drop = FALSE]

        y=as.factor(training_Grant$large_theft_rate)
        dat=data.frame(x,y)

        #train the resulting model
        if (Mod=="polynomial"){ #if our model is polynomial include degree value
          modelfit = svm(y ~ ., data = dat, kernel = Mod, cost = c, gamma = Gam,
                         degree = Deg)
        } else{modelfit = svm(y ~ ., data = dat, kernel = Mod, cost = c,
                              gamma = Gam)} #otherwise don't

        #test the resulting model and store predictions
        predictions=predict(modelfit, newdata=testing_Grant)
        perc.yes=round((length(as.numeric(predictions[predictions==1]))/
                          length(as.numeric(predictions)))*100,2)
        perc.no=100-perc.yes
        perc.preds=rbind(perc.preds,c(perc.yes,perc.no))

        #compute predictive accuracy using MCC
        score=mcc(as.numeric(predictions),testing_Grant$large_theft_rate)

        #store model
        if (Mod=="polynomial"){new_mod=c(Mod,c,Gam,Deg)}
        if (Mod!="polynomial"){new_mod=c(Mod,c,Gam,NA)}
        Modellist=rbind(Modellist,new_mod)

        #store performance score
        MCClist=c(MCClist,score)
      }
    }
  }
}

#combine models and MCC scores
results=cbind(Modellist,round(MCClist,3),perc.preds)
results=as.data.frame(results)
```

```
colnames(results)=c("Model","C","Gamma","Degree","MCC","% Yes", "% No")

#plot a histogram to visualize MCC scores
hist(as.numeric(results[,5]),xlab="prediction score",
  main="frequency of prediction \n scores Grant County")
```



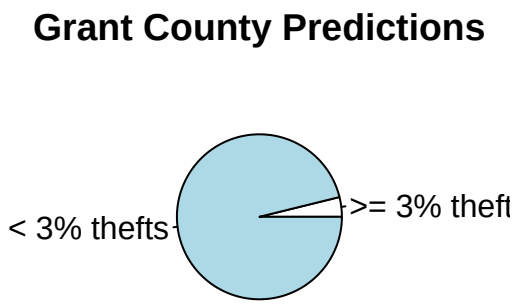
We see that our best model is around 30% accurate at making the opposite predictions as the current model. Let's define this model and again plot a pie chart of its (opposite) yes/no predictions.

```
#identify best-predicting model
best.model.index=which(abs(as.numeric(results$MCC))==
                         max(abs(as.numeric(results$MCC))))[1]
best.model=results[best.model.index,]

#visualize best models
best.model
```

```
##             Model   C Gamma Degree    MCC % Yes  % No
## new_mod.36 radial 100  0.01   <NA> -0.346  3.85 96.15
```

As shown, all of our models that predict the presence of large theft rates in Grant county, WA are non-linear functions. Now let's randomly select one of these models and observe its predictions of large theft rates in a pie chart.

```
#randomly select one model
model.number=sample(1:nrow(best.model),1)
model=best.model[model.number,]

#plot pie chart of opposite yes/no predictions
positive=as.numeric(model[1,6])
negative=as.numeric(model[1,7])
slices=c(positive,negative)
labels=c(">= 3% thefts","< 3% thefts") #opposite since MCC score<0

#output pie chart and predictive accuracy
pie(slices, labels = labels, main = "Grant County Predictions")
```



These results show that Grant county is not as likely to contain larger than 3% of the population committing thefts in a given year. We can also determine the order of influence that each predictor has on the outcome variable using variable-dropping.

Variable-Dropping: Which Variables in Our Models Impact Large Theft Rates the Most?

As previously mentioned, now that we have obtained our full model that predicts the presence of large theft rates in Grant county the best, we can determine how much each individual predictor influences the presence of Grant county large theft rates using variable-dropping. For this process, we iteratively remove one predictor from our best model at a time, compute the model's prediction score, and put the variable back. We repeat this process until all predictors have been removed from the model, and ultimately select the predictor to drop from our model that produces the highest prediction score upon removal. We now have one less predictor in our Grant county model. We then continue until we've dropped all variables from our model. The order in which each predictor is dropped determines its order of influence. For instance, the 10th predictor removed is rank 10, and has less influence over the presence of large theft rates than the 23rd predictor removed with rank 23. Let's show an example. We normalize these ranks between 0 and 1. See an exmaple.

```
predictor_rank=numeric() #initialize list to store predictor order of influence
predictor_dropped=numeric() #initialize list to store predictor dropped

#store parameters for best model from grid search
best.model.kernel=model[1,1];best.model.C=as.numeric(model[1,2])
best.model.gamma=as.numeric(model[1,3])
if (best.model.kernel=="polynomial"){best.model.degree=as.numeric(model[1,4])}

total_num_preds=length(names(dat))-1 #initialize total number predictors to start

rank=0 #initialize rank

while (length(predictor_dropped)!=total_num_preds-1){

predictor_removed=numeric() #initialize list to store predictor removed
#list of predictors in Grant county data
```

```
predictors=names(dat)[-which(names(dat)=="y")]

#intiialize list to store prediction scores after each predictor is removed
MCClist=numeric()

for (i in 1:(length(predictors)-1)){
  predictor.i=predictors[i] #for each predictor
  predictor.i.vals=dat[which(names(dat)==predictor.i)]#compute its vals

  #store predictor removed and its vals in data frame
  predictor.i.df=as.data.frame(predictor.i.vals)
  colnames(predictor.i.df)=predictor.i

  predictor_removed=c(predictor_removed,predictor.i) #add this predictor to list
  dat=dat[,-which(names(dat)==predictor.i)] #remove this predictor from training
    #data

  #train the model
  if (best.model.kernel=="polynomial"){modelfit = svm(y ~ ., data = dat,
                kernel = best.model.kernel, cost = best.model.C,
                gamma = best.model.gamma, degree = best.model.degree)
  } else{modelfit = svm(y ~ ., data = dat, kernel = best.model.kernel,
                cost = best.model.C,
                gamma = best.model.gamma)}

  #test the model
  predictions=predict(modelfit, newdata=testing_Grant)

  #compute model's performance
  score=mcc(as.numeric(predictions),testing_Grant$large_theft_rate)

  #store
  MCClist=c(MCClist,score)

  #put the predictor back
  dat=cbind(dat,predictor.i.df)
}

#add last remaining predictor and its score to list
predictor_removed=c(predictor_removed,predictors[length(predictors)])
MCClist=c(MCClist,0)

#identify predictor that produced the max score when removed
MCC.max.index=which(abs(MCClist)==max(abs(MCClist)))[1]
pred.max.MCC=predictor_removed[MCC.max.index]

#drop this predictor from the model
dat=dat[,-which(names(dat)==pred.max.MCC)]
predictor_dropped=c(predictor_dropped,pred.max.MCC)

#store its rank
rank=rank+1
predictor_rank=c(predictor_rank,rank)
```

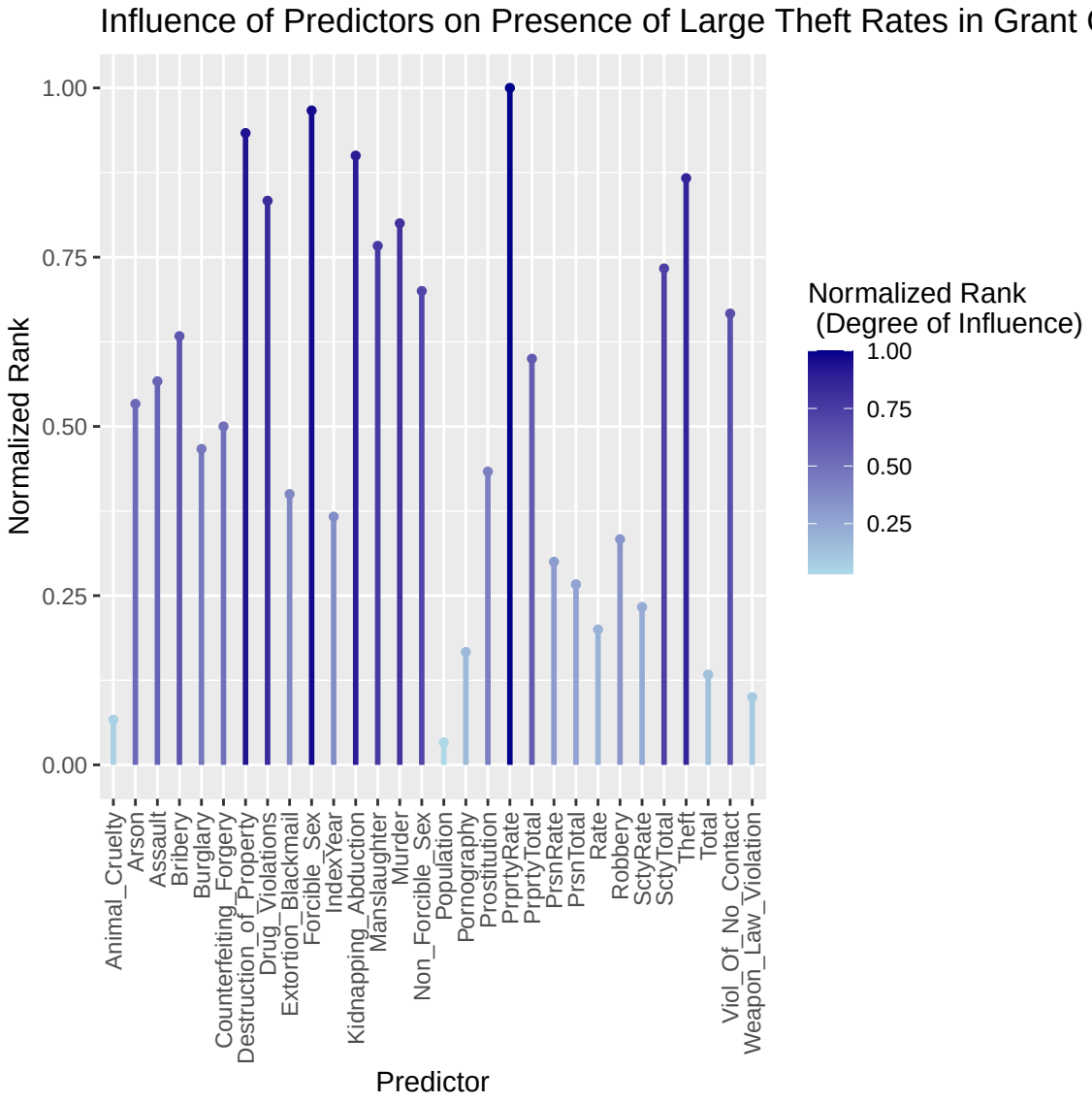
```
}

#store the last predictor left in the model
predictor_dropped=c(predictor_dropped,setdiff(predictors,predictor_dropped))

#store last predictor's rank
rank=rank+1
predictor_rank=c(predictor_rank,rank)

#output ranking table
rank_table=cbind(predictor_dropped,predictor_rank)
rank_table=as.data.frame(rank_table);colnames(rank_table)=
  c("Predictor Removed","Predictor Rank")

#normalize ranks and add to rank table
normalized_ranks=as.numeric(rank_table$`Predictor Rank`)/total_num_preds
normalized_ranks=as.data.frame(normalized_ranks)
colnames(normalized_ranks)="Normalized Rank"
rank_table=cbind(rank_table,normalized_ranks)

rank_table
```

```
##          Predictor Removed Predictor Rank Normalized Rank
## 1               Population              1      0.03333333
## 2           Animal_Cruelty              2      0.06666667
## 3     Weapon_Law_Violation              3      0.10000000
## 4                    Total              4      0.13333333
## 5              Pornography              5      0.16666667
## 6                     Rate              6      0.20000000
## 7                 SctyRate              7      0.23333333
## 8                PrsnTotal              8      0.26666667
## 9                 PrsnRate              9      0.30000000
## 10                Robbery             10      0.33333333
## 11              IndexYear             11      0.36666667
## 12     Extortion_Blackmail             12      0.40000000
## 13            Prostitution             13      0.43333333
## 14                Burglary             14      0.46666667
## 15  Counterfeiting_Forgery             15      0.50000000
## 16                  Arson             16      0.53333333
## 17                Assault             17      0.56666667
## 18             PrprtyTotal             18      0.60000000
## 19                 Bribery             19      0.63333333
## 20      Viol_Of_No_Contact             20      0.66666667
## 21        Non_Forcible_Sex             21      0.70000000
## 22               SctyTotal             22      0.73333333
## 23            Manslaughter             23      0.76666667
## 24                  Murder             24      0.80000000
## 25         Drug_Violations             25      0.83333333
## 26                   Theft             26      0.86666667
## 27    Kidnapping_Abduction             27      0.90000000
## 28 Destruction_of_Property             28      0.93333333
## 29            Forcible_Sex             29      0.96666667
## 30              PrprtyRate             30      1.00000000
```

Visualizing the Influence of Predictors on the Prescense of Large Theft Rates in Grant County, WA

We can visualize the order of influence (rankings) of predictors on theft rates in Grant county using a heat map. Recall that our best non-linear model predicts that there will be less than 3% of crimes committed in Grant county. Hence, the taller/darker a predictor's line, the higher its influence is over the non-existence of large theft rates in Grant county. Vice versa for shorter/lighter lines.

```
#set x-axis of predictors for heat map
x=setdiff(names(training_Grant),"large_theft_rate")
#set y-axis of normalized ranks for heat map
y=numeric()
for (i in 1:length(x)){
  pred=x[i] #define predictor
  pred.loc=which(rank_table[,1]==pred) #find its row number in our rank table
  pred.rank=rank_table[pred.loc,3] #define its normalized rank in rank table
  y=c(y,pred.rank) #add its rank to list
}

#store (x,y) points for heat map in data frame
data=cbind(x,1:total_num_preds,y)
data=as.data.frame(data);colnames(data)=
  c("Predictor","Predictor ID","Normalized Rank")


ggplot(data, aes(x = x, y = y)) +
  geom_segment(aes(xend = Predictor, yend = 0, color = y),
               size = 1) +  #add lines
  geom_point(shape = 16, aes(color = y)) +  #plot (x,y) points
  labs(x = "Predictor", y = "Normalized Rank",
  title =
  "Influence of Predictors on Presence of Large Theft Rates in Grant County") +
  scale_color_gradient(low = "lightblue", high = "darkblue",
                       name = "Normalized Rank \n (Degree of Influence)") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



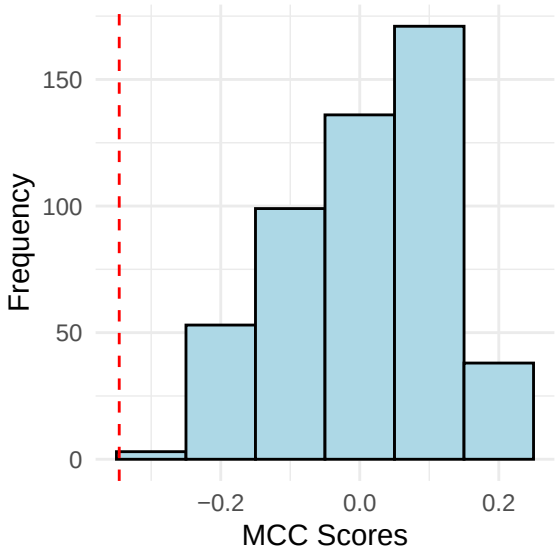
As we can see, destruction of property, drug violations, forcible sex, kidnapping/abduction, property rates, and theft are all factors that have high influence over the absence of large theft rates in Grant county, WA.

Assessing the Significance of Model Performance

It is important to check that our best model's MCC score is statistically significant. That is, we want to assure ourselves that this prediction measure didn't just occur by random chance. In order to do this, we perform a statistical test called a permutation test. The way this test works is, we randomly shuffle our "large_theft_rate" responses for Grant county, and computed the resulting MCC score after this permutation, called a permuted MCC score. We then plot a histogram of the frequency of all permuted MCC scores along with a vertical line at our model's originally observed score "best.model.performance" (coded earlier). The idea is this - with a positive MCC score, if most our histogram falls below (to the left of) our original observed score, this means that a majority of our randomly-occurring MCC scores are NOT at least as good as our observed score. On the other hand, with a negative MCC score, if most of our histogram falls above (to the right of) our original observed score, this means that a majority of our randomly-occurring MCC scores are NOT at least as good as our observed score. In either case, this would indicate that our MCC score is statistically significant (did not occur by random chance). This step is important, as we want to ensure that

our model's performance is good.

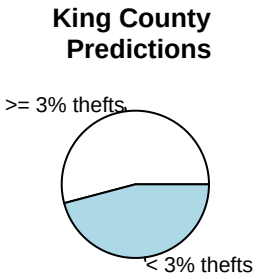
```
#observed MCC score
observed_MCC=as.numeric(best.model[1,5])

#compute permuted MCC scores
permuted_MCC=numeric()

for (i in 1:500){

  #shuffle the responses
  training_Grant.shuffled_responses=sample(training_Grant$large_theft_rate)

  #define the Grant county training data with shuffled responses
  x=training_Grant[, -which(names(training_Grant)=="large_theft_rate"),
                                                        drop = FALSE]

  y=as.factor(training_Grant.shuffled_responses)
  dat=data.frame(x,y)

  #train the model with shuffled responses
  if (best.model.kernel=="polynomial"){modelfit = svm(y ~ ., data = dat,
                kernel = best.model.kernel, cost = best.model.C,
                gamma = best.model.gamma, degree = best.model.degree)
  } else{modelfit = svm(y ~ ., data = dat, kernel = best.model.kernel,
                cost = best.model.C,
                gamma = best.model.gamma)}

  #test the model
  predictions=predict(modelfit, newdata=testing_Grant)

  #compute model's performance
  score=mcc(as.numeric(predictions),testing_Grant$large_theft_rate)

  #store
  permuted_MCC[i]=score
}

histogram.df=as.data.frame(permuted_MCC)

ggplot(histogram.df, aes(x = permuted_MCC)) +
  geom_histogram(binwidth = 0.1, fill = "lightblue", color = "black") +
  geom_vline(xintercept = observed_MCC, linetype = "dashed", color = "red") +
  labs(x = "MCC Scores", y = "Frequency") +
  theme_minimal()
```

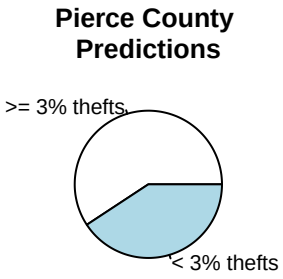


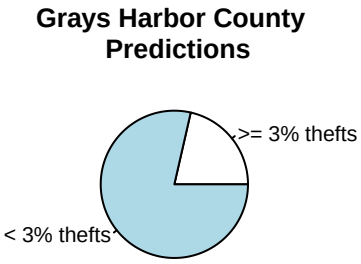
Hence, by our permutation test, our best model's predictive accuracy is statistically significant (i.e did not occur by chance).

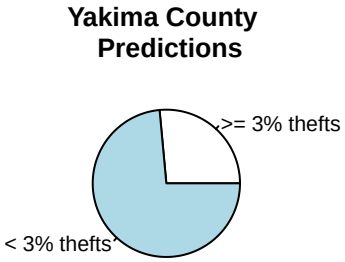
Comparing which counties are most likely to contain high crime rates in Grant county

We can now use grid search cross-validation to identify SVM models that predict the presence of large theft rates in our other four counties of interest (Grays Harbor, Pierce, King, Yakima). Then, we can display these model predictions as pie charts, and compare the results we get to determine which county is safest to live in theft-wise.









We can similarly compare the best models' performance scores and in each county also.

```
##            Grant Grays Harbor Pierce   King Yakima
## MCC score -0.346       -0.352 -0.464 -0.476 -0.381
```

Answering Our Question: Which Counties in WA are Most Likely to Contain Low Crime?

Based on our results, we can see that the county that is most likely to contain low theft rates is Grays Harbor County. When deciding which county to live in, one could use our results to select Grays.