Regression with SVM

Zachary Canoot [*]Gray Simpson [†]

23 October, 2022

SVM Regression

Support Vector Machines can divide data into classes by a hyperplane in multidimensional space. This line separates classes by finding minimum distance of margins between support vectors. Once we calculate support vectors for our model (given an input of slack in the margins optimized with validation data), we can then classify the data in relation to the margins on the hyperplane.

In the case of Regression, we apply this logic to fit a line to the data (as opposed to divide the data). Classification minimizes the margins such that all examples on either side of the margins are assumed to be classified correctly. SVM Regression's minimization function will instead find a hyper plane that fits the data within a certain accuracy. Specifically, the support vectors have the largest amount of error (distance) from the hyperplane, and everything within those support vectors (the margin) is assumed correctly fitted. Thus the hyperplane fits the data like simple regression.

We are going to apply this algorithm to a simple 1 target, 1 predictor data set, and get a nice visual demonstration of the hyperplane.

Exploring our Data

Our data, found on Kaggle is a very detailed and large collection of samples of larval fish data. However, I'm only interested in the temperature of the ocean water in relation to it's salinity. So lets read it in and trim it down to just the necessary columns and a handleable training size. After finishing the document, I realized I had to trim the file size for github, so note that the actual data on kaggle is much larger!

> Note: We are going to sample the data to a smaller size right now, knowing that the last slide for SVM warned this may be neccessary

We think it is worth leaving depth in, sense this will probably be a good predictor of temperature, if not as easy to see visually.

> I'm leaving in the 2 code chunks below as comments, but they are referring to data *before* I trimmed down the csv file to upload to github!

```
# This takes a second
# ocean_data <- read.csv("bottle.csv")
```

[*]Zaiquiri's Portfolio
[†]Gray's Porfolio

```r
# Selecting the wanted columns: 5= T_degC, 6=Depthm, and 7=Salnty
# df <- ocean_data[c(5,6,7)]
# Removing NAs
# which_nas <- apply(df, 1, function(X) any(is.na(X)))
# nas <- length(which(which_nas))
# size <- nrow(df)
# ratio <- nas/size
# size <- format(size, big.mark = ",", scientific = FALSE)
# nas <- format(length(which(which_nas)), big.mark = ",", scientific = FALSE)
# sprintf("%%%.2f of the %s large dataset contains NA's. Removing %s", ratio*100, size, nas)
# df <- na.omit(df)
```

864,863 rows! That's still quite large! Lets reduce to exactly 10,000

```r
# set.seed(8)
# df <- df[sample(1:nrow(df),10000,replace=FALSE),]
# nrow(df)
# head(df)
```

```r
# write.csv(df,"ocean_data.csv")
```

We can then split into training, testing, and validation data

```r
# The line below reads in the modified csv
set.seed(8)
df <- read.csv("ocean_data.csv")
spec <- c(train=.6, test=.2, validate=.2)
i <- sample(cut(1:nrow(df), nrow(df)*cumsum(c(0,spec)), labels=names(spec)))
train <- df[i=="train",]
test <- df[i=="test",]
vald <- df[i=="validate",]
```

Graphical and Text Analysis

Lets look at the data we are using to build our models:

```r
summary(train)
```

```
##        X              Depthm           T_degC          Salnty
##  Min.   :    78   Min.   :   0.0   Min.   : 1.49   Min.   :31.44
##  1st Qu.:242486   1st Qu.:  48.0   1st Qu.: 7.80   1st Qu.:33.49
##  Median :454128   Median : 125.0   Median :10.08   Median :33.87
##  Mean   :448242   Mean   : 224.8   Mean   :10.81   Mean   :33.84
##  3rd Qu.:660431   3rd Qu.: 300.0   3rd Qu.:13.80   3rd Qu.:34.20
##  Max.   :864738   Max.   :4761.0   Max.   :28.82   Max.   :35.51
```

Some things to note:

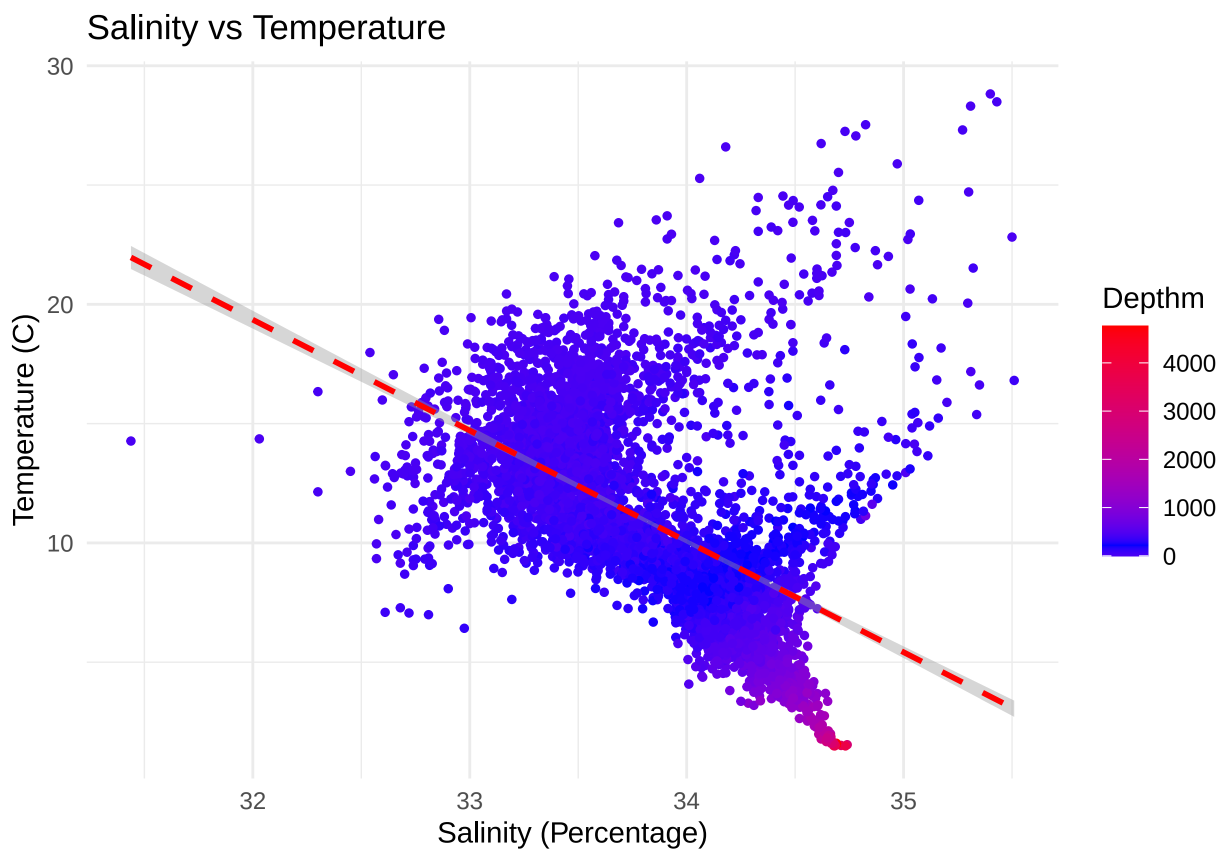
1. A min depth of 0 is odd, but makes sense as a surface reading
2. The outliers don't seem that far, so this might be some good data to look at

3. From the mean values we can gather our average case: At a depth of of 228 meters, the temperature was ~11 degrees Celsius (51.8 Fahrenheit), with a Salinity of ~34 percent, which matches averages easily found online.
4. There weren't that many data reads in extremely salty waters (>36%)

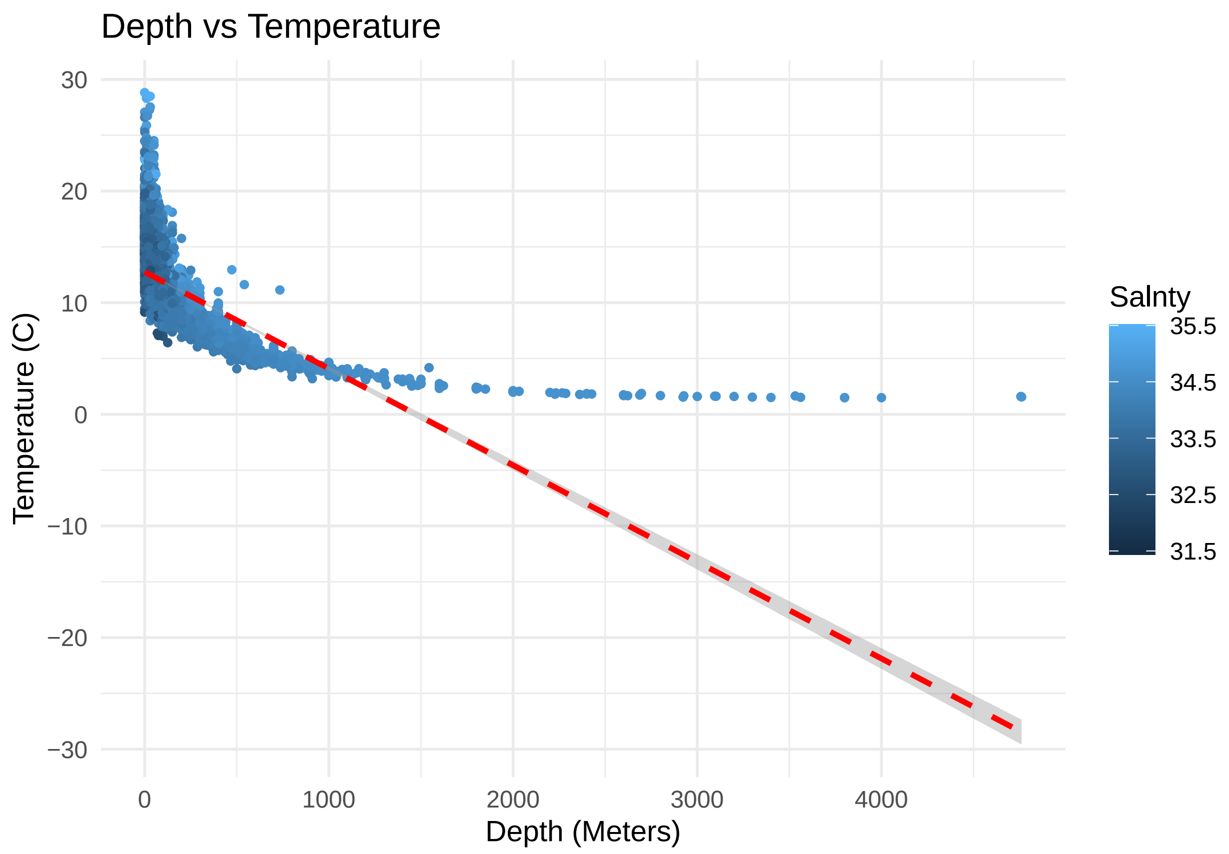
I want to graph the correlation from salinity to temperature, a 3d graph of depth, temperature, and salinity, etc.

```r
library(ggplot2)
theme_set(theme_minimal())
mid <- mean(train$Depthm)
ggplot(train, aes(x=Salnty, y=T_degC)) +
  geom_point(pch=19, aes(color = Depthm), size=1) +
  geom_smooth(formula="y~x", method="lm", color="red", linetype=2) +
  labs(title="Salinity vs Temperature", x="Salinity (Percentage)", y
       ="Temperature (C)") +
  scale_color_gradient2(midpoint=mid, low = "red", mid = "blue", high = "red", space = "lab")
```

```
## Warning: The `space` argument of `gradient_n_pal()` only supports be "Lab" as of scales
## 0.3.0.
## i The deprecated feature was likely used in the scales package.
##   Please report the issue at <https://github.com/r-lib/scales/issues>.
```



```
ggplot(train, aes(x=Depthm, y=T_degC)) +
  geom_point(pch=19, aes(color = Salnty), size=1) +
  geom_smooth(formula="y~x", method="lm", color="red", linetype=2) +
  labs(title="Depth vs Temperature", x="Depth (Meters)", y
      ="Temperature (C)")
```



We can see by this scatter plot, along with the convenient smoothing line that there is definitely a trend or correlation in the data. The smoothing line shows what the result of a linear regression might be on the model, and it's confidence interval (indicated by the gray border on the line) indicates the model... holds some water.

Both Depth and Salinity have a strong correlation, but it seems Depth might have an exponential decay like relationship with temperature. That makes me wonder how the best way to portray these relationships.

```
cor(df)
```

```
##                  X     Depthm      T_degC     Salnty
## X       1.00000000 -0.1430773  0.06834582 -0.1942345
## Depthm -0.14307734  1.0000000 -0.67392480  0.5760050
## T_degC  0.06834582 -0.6739248  1.00000000 -0.5096666
## Salnty -0.19423450  0.5760050 -0.50966658  1.0000000
```

Looking at the correlation as well as our graphs, simple visual analysis tells us that both depth and salinity are useful predictors of temperature, and a linear regression model would be useful. However, the relationships are complex, and we may find that a SVM regression model with a polynomial kernel might yield a good result. This is because it would be able to mold together our 2 predictors.

Lets have a multiple-predictor linear regression model to compare our SVM results to and then carry on.

```
# Creating a lm using both predictors and graphing predictions
lm <- lm(T_degC ~ Salnty + Depthm, train)
pred <- predict(lm, newdata=test)
preddf <- data.frame(temp_pred=pred, salinity=test$Salnty)
summary(lm)
```

```
##
## Call:
## lm(formula = T_degC ~ Salnty + Depthm, data = train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -6.991 -2.080 -1.022  1.270 24.951
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 72.8185801  3.4892832   20.87   <2e-16 ***
## Salnty      -1.7846393  0.1036633  -17.22   <2e-16 ***
## Depthm      -0.0072022  0.0001485  -48.49   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.047 on 5997 degrees of freedom
## Multiple R-squared:  0.4702, Adjusted R-squared:   0.47
## F-statistic:  2661 on 2 and 5997 DF,  p-value: < 2.2e-16
```

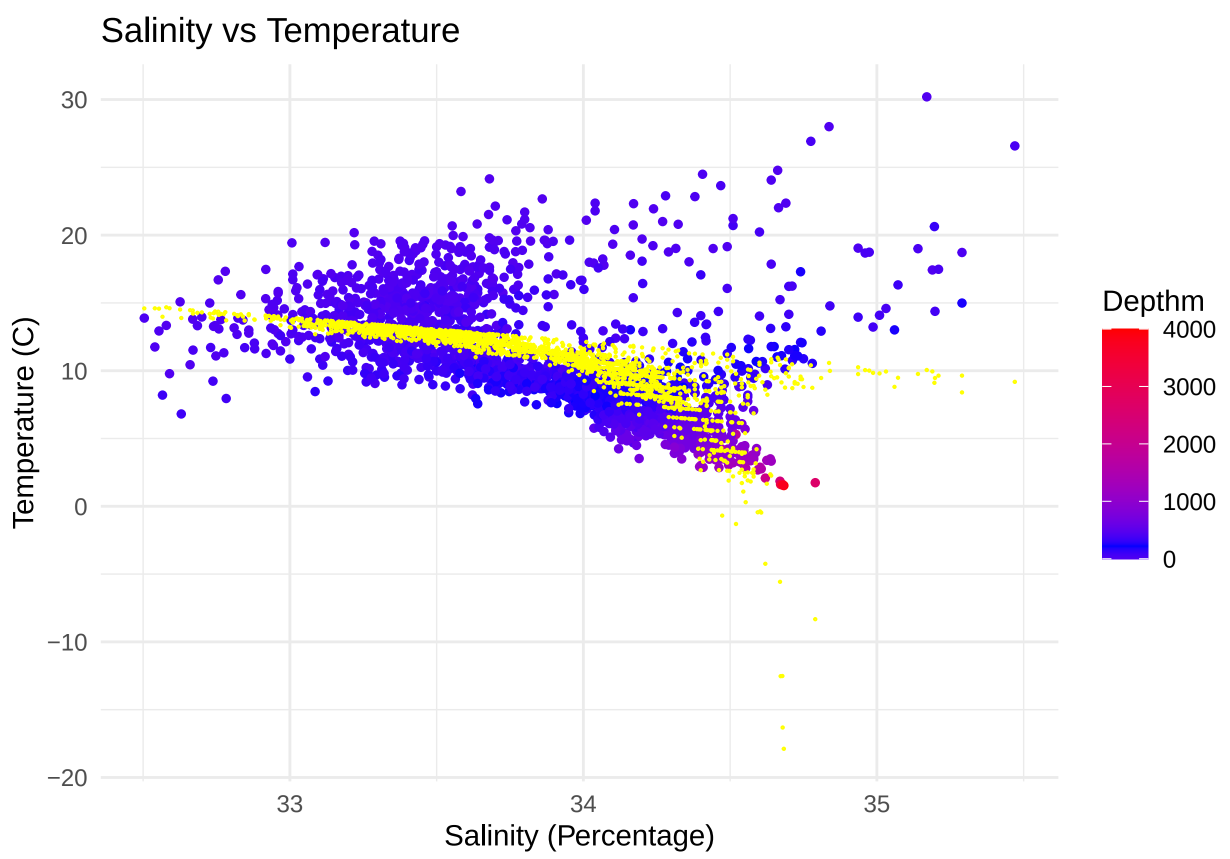
```
cor <-cor(pred, test$T_degC)
mse <- mean((pred-test$T_degC)^2)
sprintf("Correlation of our prediction: %s", cor)
```

```
## [1] "Correlation of our prediction: 0.69737972427243"
```

```
sprintf("Mean Squared Error: %s", mse)
```

```
## [1] "Mean Squared Error: 9.39531280700042"
```

```
ggplot(test, aes(x=Salnty, y=T_degC)) +
  geom_point(pch=19, aes(color = Depthm), size=1) +
  geom_point(pch=1, color="yellow",data=preddf, aes(x=salinity, y=temp_pred), size=.1) +
  labs(title="Salinity vs Temperature", x="Salinity (Percentage)", y
       ="Temperature (C)") +
  scale_color_gradient2(midpoint=mid, low = "red", mid = "blue", high = "red", space = "lab")
```



Performing SVM Regression

Lets just create a model for each type, tune the hyperparameters, and analyze the results.

Linear Kernel

```r
library(e1071)
svmlin <- svm(T_degC~Depthm+Salnty, data=train, kernel="linear", cost=10, scale=TRUE)
summary(svmlin)
```

```
##
## Call:
## svm(formula = T_degC ~ Depthm + Salnty, data = train, kernel = "linear",
##     cost = 10, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.5
##     epsilon:  0.1
##
```

```
##
## Number of Support Vectors:  5060
```

```
pred <- predict(svmlin, newdata=test)
cor_svmlin <- cor(pred, test$T_degC)
mse_svmlin <- mean((pred - test$T_degC)^2)
sprintf("Correlation of our prediction: %s", cor_svmlin)
```

```
## [1] "Correlation of our prediction: 0.686181141980326"
```

```
sprintf("Mean Squared Error: %s", mse_svmlin)
```

```
## [1] "Mean Squared Error: 10.7137242527504"
```

That is a little worse then our baseline, lets try and tune it using a smaller subsample

```
# Just using the ranges used in examples
tune_svmlin <- tune(svm, T_degC ~ Depthm + Salnty, data = vald, kernel="linear", ranges=list(cost=c(0.0
tune_svmlin$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = T_degC ~ Depthm + Salnty, data = vald,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  5
##       gamma:  0.5
##     epsilon:  0.1
##
##
## Number of Support Vectors:  1710
```

We ran it with 1000 rows, and got a cost value of 1, then ran it with 3000 and got 100, then ran it with 5000, and then decided to wait for the 10000 to finish and got.. well 100. This suggest I should increase the range, but also that the linear model isn't the best solution to our data, considering the cost had to be so high.

```
svmlin <- svm(T_degC~Depthm+Salnty, data=train, kernel="linear", cost=5, scale=TRUE)
pred <- predict(svmlin, newdata=test)
tuned_cor_svmlin <- cor(pred, test$T_degC)
tuned_mse_svmlin <- mean((pred - test$T_degC)^2)
sprintf("Correlation of our prediction: %s", tuned_cor_svmlin)
```

```
## [1] "Correlation of our prediction: 0.686159913440689"
```

```
sprintf("Mean Squared Error: %s", tuned_mse_svmlin)
```

```
## [1] "Mean Squared Error: 10.7156051065854"
```

The error was higher, and the correlation was lower so we didn't improve. Tuning also barely effected the results!

Polynomial Kernel

```
svmpoly <- svm(T_degC~Depthm+Salnty, data=train, kernel="polynomial", cost=5, scale=TRUE)
summary(svmpoly)
```

```
##
## Call:
## svm(formula = T_degC ~ Depthm + Salnty, data = train, kernel = "polynomial",
##     cost = 5, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  polynomial
##        cost:  5
##      degree:  3
##       gamma:  0.5
##      coef.0:  0
##     epsilon:  0.1
##
##
## Number of Support Vectors:  5219
```

```
pred <- predict(svmpoly, newdata=test)
cor_svmpoly <- cor(pred, test$T_degC)
mse_svmpoly <- mean((pred - test$T_degC)^2)
sprintf("Correlation of our prediction: %s", cor_svmpoly)
```

```
## [1] "Correlation of our prediction: 0.617921033892596"
```

```
sprintf("Mean Squared Error: %s", mse_svmpoly)
```

```
## [1] "Mean Squared Error: 11.9707882748739"
```

Well that didn't go well! MSE is even higher then the baseline, so I can try to tune

```
# tune_svmpoly <- tune(svm, T_degC ~ Depthm + Salnty, data = vald, kernel="polynomial", ranges=list(cos
# tune_svmlin$best.model
```

This would take a while to run, but would simply keep trying to increase the cost. But then that produces a model that over fits the training data! It seems this simply isn't the best way to fit this data and is a bit worse then simple regression.

Radial Kernel

My only hope...

```
svmrad <- svm(T_degC~Depthm+Salnty, data=train, kernel="radial", cost=5, gamma=.5, scale=TRUE)
summary(svmrad)
```

```
##
## Call:
## svm(formula = T_degC ~ Depthm + Salnty, data = train, kernel = "radial",
##     cost = 5, gamma = 0.5, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  5
##       gamma:  0.5
##     epsilon:  0.1
##
##
## Number of Support Vectors:  3628
```

```
pred <- predict(svmrad, newdata=test)
cor_svmrad <- cor(pred, test$T_degC)
mse_svmrad <- mean((pred - test$T_degC)^2)
sprintf("Correlation of our prediction: %s", cor_svmrad)
```

```
## [1] "Correlation of our prediction: 0.919869113848864"
```

```
sprintf("Mean Squared Error: %s", mse_svmrad)
```

```
## [1] "Mean Squared Error: 2.87362504846235"
```

WOAH. What a great result! That is a good correlation, and what seems to be a pretty good MSE, and both are much better then our baseline linear regression. Lets try and tune

```
tune_svmrad <- tune(svm, T_degC ~ Depthm + Salnty, data = vald, kernel="radial", ranges=list(cost=c(0.0
tune_svmrad$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = T_degC ~ Depthm + Salnty, data = vald,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100), gamma = c(0.5,
##         1, 2, 3, 4)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.5
##     epsilon:  0.1
##
##
## Number of Support Vectors:  1255
```

```
svmrad <- svm(T_degC~Depthm+Salnty, data=train, kernel="radial", cost=10, gamma=1, scale=TRUE)
pred <- predict(svmrad, newdata=test)
cor_svmrad <- cor(pred, test$T_degC)
mse_svmrad <- mean((pred - test$T_degC)^2)
sprintf("Correlation of our prediction: %s", cor_svmrad)
```

```
## [1] "Correlation of our prediction: 0.921115235162866"
```

```
sprintf("Mean Squared Error: %s", mse_svmrad)
```

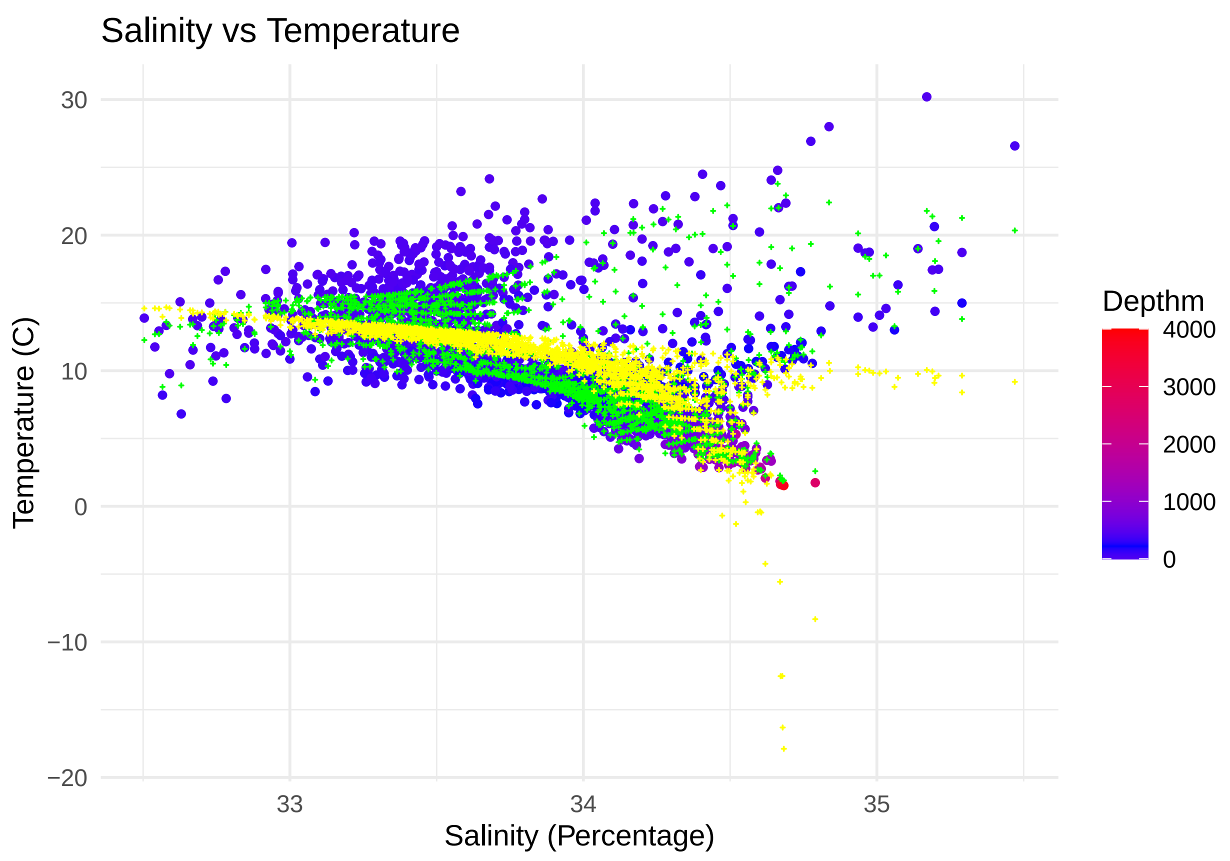
```
## [1] "Mean Squared Error: 2.82336703728242"
```

It made the results a bit marginally worse, which suggests over fitting of the data with the training data vs the validation data.

Either way, good results! Lets graph it!

```
preddf2 <- data.frame(temp_pred=pred, salinity=test$Salnty)

ggplot(test, aes(x=Salnty, y=T_degC)) +
  geom_point(pch=19, aes(color = Depthm), size=1) +
  geom_point(pch=3, color="green",data=preddf2, aes(x=salinity, y=temp_pred), size=.2) +
  geom_point(pch=3, color="yellow",data=preddf, aes(x=salinity, y=temp_pred), size=.2) +
  labs(title="Salinity vs Temperature", x="Salinity (Percentage)", y
       ="Temperature (C)") +
  scale_color_gradient2(midpoint=mid, low = "red", mid = "blue", high = "red", space = "lab")
```



The green points indicate our new Just a bit lower then our linear regression model at the start, which suggests less susceptibility to outliers.

Analysis

I would love to visualize the radial kernel to get an idea for how it works on this data, but quoting StatQuest: "Because the Radial Kernel finds Support Vector Classifiers in infinite dimensions, it's not possible to visualize what it does". While I know that you can still get an approximation, I'm going to go ahead and skip figuring out how to do that for now.

The main topic to analyze here is why the radial kernel was better than the polynomial or the linear kernel. Each kernel represents a different method of transforming the data so that a hyperplane may divide the data, or in this case, fit a function.

- The linear kernel is simple, it fits a hyperplane to the data
- The polynomial kernel transforms the data in such a way to mimic adding more features to the data set, really just by mapping the input data to a polynomial of a higher degree. By mapping values in a higher degree space, say, to the second degree, what really is a circular data set classification can now have a straight line drawn through it.
- The radial kernel compares the distance between every 2 values in the input data, and scales the data by the value of it's distance. This mimics nearest neighbor, where the model predicts every value with increasing weight supplied to its neighbors. The kernel can then map the input to a higher (infinite) dimensional space where it is easiest to fit a hyperplane that best maximizes the margins of the model… it's not exactly easy to wrap a brain around

Because we found in our initial analysis that the relationship between our two predictors and our target was a complicated combination of a a not-so-linear relationship and a perhaps-exponential relationship, it makes sense that a complicated model like SVM with a radial kernel would have been able to find better results

To think of this in terms of the data, the radial kernel was best able to interpret how depth related to temperature compared to salinity. As depth increased, it got colder, and salinity mattered less. The interactions between the data were important in this data! So while linear regression was good at approximating the relationship, the radial kernel was *probably* able to better fit the data.

Link Dump!

https://stackoverflow.com/questions/13353213/gradient-of-n-colors-ranging-from-color-1-and-color-2

https://statisticsglobe.com/sprintf-r-function-example

https://statisticsglobe.com/display-large-numbers-separated-with-comma-in-r

I referenced the book for a lot of ggplot2 stuff, as well as code on SVM.

https://www.youtube.com/watch?v=OpPBhBQA7fE&list=PLfe6IcA_dEWkcHFfBA6XSXW31H8t4XSbB&index=16

https://stackoverflow.com/questions/37329074/geom-smooth-and-exponential-fits

https://machinelearningmastery.com/regression-metrics-for-machine-learning/

https://www.youtube.com/watch?v=Qc5IyLW_hns