Predicting Diabetes with Support Vector Machines

Cliff Lee

2023-11-24

Abstract

Both random forests (RF) and support vector machines (SVM) are two of the easiest and most import algorithms that offer excellent prediction for relatively little effort. Howevever, their methods are quite different and so it's important to know their implementation, requirements and limitations to use them effectively.

When using an SVM with the previous diabetes dataset, we found both SVM and RF models have roughly equivalent accuracy rates of 88%.

Generally from the few articles, SVMs are preferable. Even though both RFs and SVMs perform well, SVM can do perform regression analysis; RFs cannot. Imagine a decision tree that somehow gave an exact or rough number, the number of levels would be incredibly high.

The other conclusions regarding SVMs is you'll want to inspect your dataset to make sure you have enough data and not too many features (features < observations). You'll also want to reduce bias by normalizing the continous features; i.e. scale all variables the same way. Also, you'll want to visually inspect your data to see if the features are visually separable. If they are, then great. If not (this occurs with real data) then you should tune the hyperparameters (e.g. C) and then select your kernel.

Also, using v-fold cross validation and grid-search for tuning hyperparameters.

Literature

Researching through articles and studies regarding the use of SVMs shows their many different uses: covid detection; wine quality[1]; insurance reimbursement[2]; social media sentiment analysis. Researchers found SVMs perform well in many circumstances, however they also emphasize you cannot avoid the crucial steps of exploring and preprocessing the dataset, checking assumptions and comparing against other models [3]. No algorithm can be expected to outperform all others for every problem (according to the 'no free lunch' theorem of algorithms).

Also, you may not need all possible features and for performance reasons, you can choose a subset of them [4][5]; SVMs perform well, when the dataset has either dramatically more features than observations or vice versa. SVMs with both large features and observations suffer from poor training performance as every datapoint (or vector) has to be checked as a possible support candidate. For large datasets, the amount of required memory and calculations can make training times unfortunately slow. In the articles referenced, SVMs along with RFs typically performed with the highest accuracy rate. Partially because SVMs tend to behave like K-nearest neighbor models and they are non-parametric.

Aside from classifications, SVMs can be used for regression analysis as well. The idea being the hyperplane serves as a regression for a given dataset. As with classification, SVM regression models still require preprocessing and proper feature handling but they are robust to outliers and have very good generalization

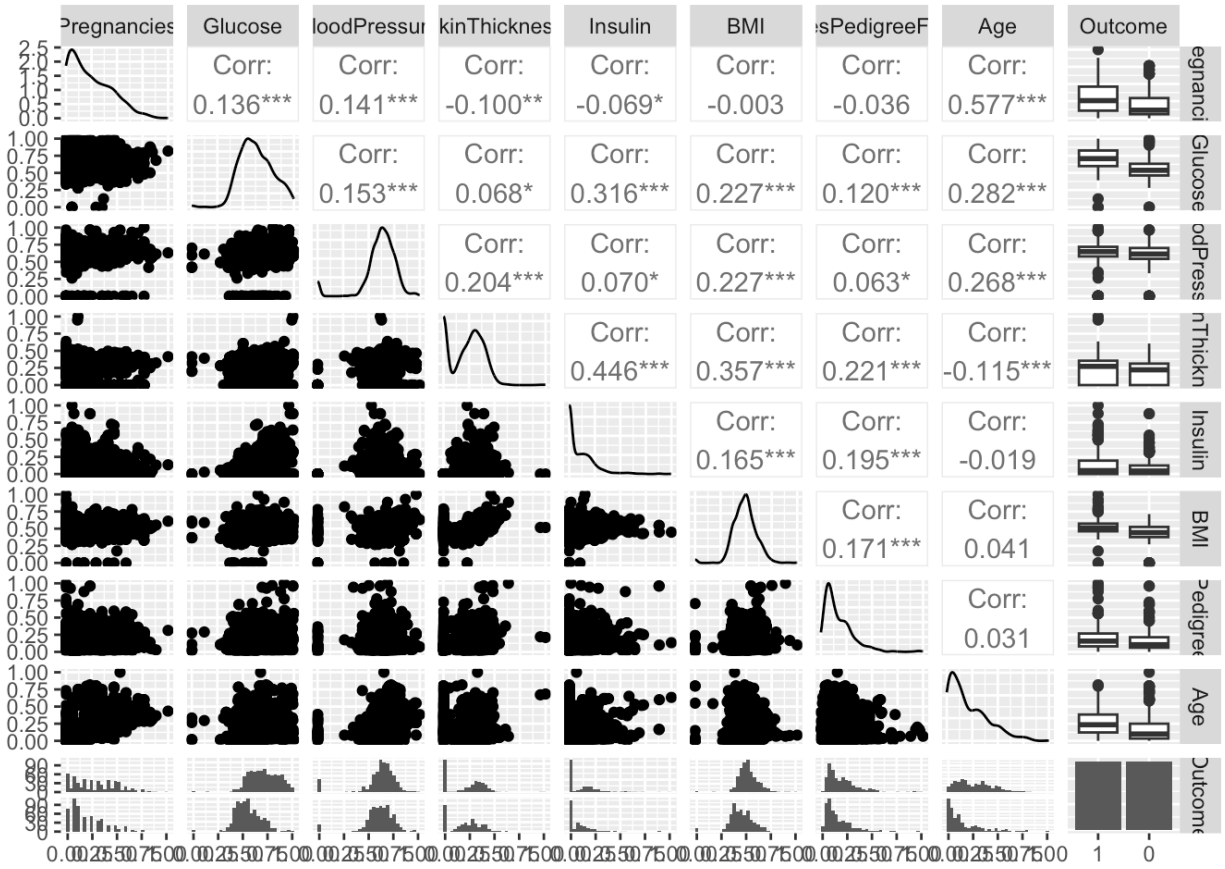
and prediction capability. However, with large datasets or noisy data, SVMs may not perform well (as with other models).

Lastly, at least a few articles highlighted general best practices that is common to all algorithms: using all the data as much as possible but scaling all the continous features to reduce bias. Also, using v-fold and grid-search libraries to tune the various hyperparamters for the various SVM kernels helped to maximize overall accuracy.

Data Exploration

Reusing the previous diabetees dataset, we have eight continuous independent variable and one dependant category (outcome). An outcome value of zero means a patients does not have diabetes, a value of one means she/he does. Again, the SMOTE library balanced the dataset for the dependent variable. The difference in this assignment follows the advice of the reading material; I have scaled all of the predictors so they vary from zero to one.

```
##   Pregnancies         Glucose       BloodPressure    SkinThickness
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.05882   1st Qu.:0.5176   1st Qu.:0.5614   1st Qu.:0.0000
##  Median :0.17647   Median :0.6229   Median :0.6316   Median :0.2506
##  Mean   :0.24209   Mean   :0.6330   Mean   :0.6149   Mean   :0.2146
##  3rd Qu.:0.39437   3rd Qu.:0.7407   3rd Qu.:0.7018   3rd Qu.:0.3434
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##     Insulin             BMI         DiabetesPedigreeFunction      Age
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.00000          Min.   :0.00000
##  1st Qu.:0.00000   1st Qu.:0.4173   1st Qu.:0.07192          1st Qu.:0.05265
##  Median :0.04551   Median :0.4866   Median :0.12757          Median :0.15000
##  Mean   :0.09795   Mean   :0.4864   Mean   :0.17195          Mean   :0.21139
##  3rd Qu.:0.16548   3rd Qu.:0.5499   3rd Qu.:0.24101          3rd Qu.:0.33333
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.00000          Max.   :1.00000
##  Outcome
##  1:536
##  0:536
##
##
##
##
```

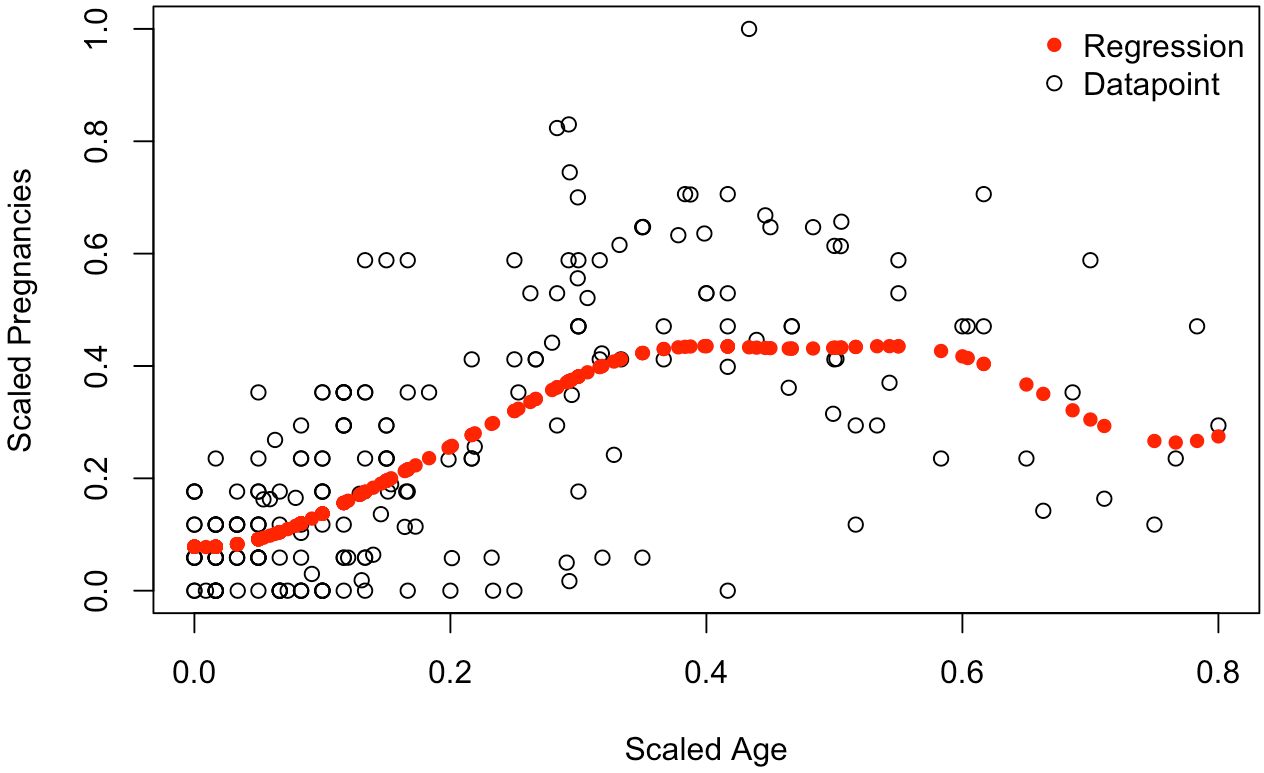



```
set.seed(1234)

sample_set <- sample(nrow(data), round(nrow(data)*0.80), replace = FALSE)

data_train <- data[sample_set,]
data_test <- data[-sample_set,]
```

SVM Regression

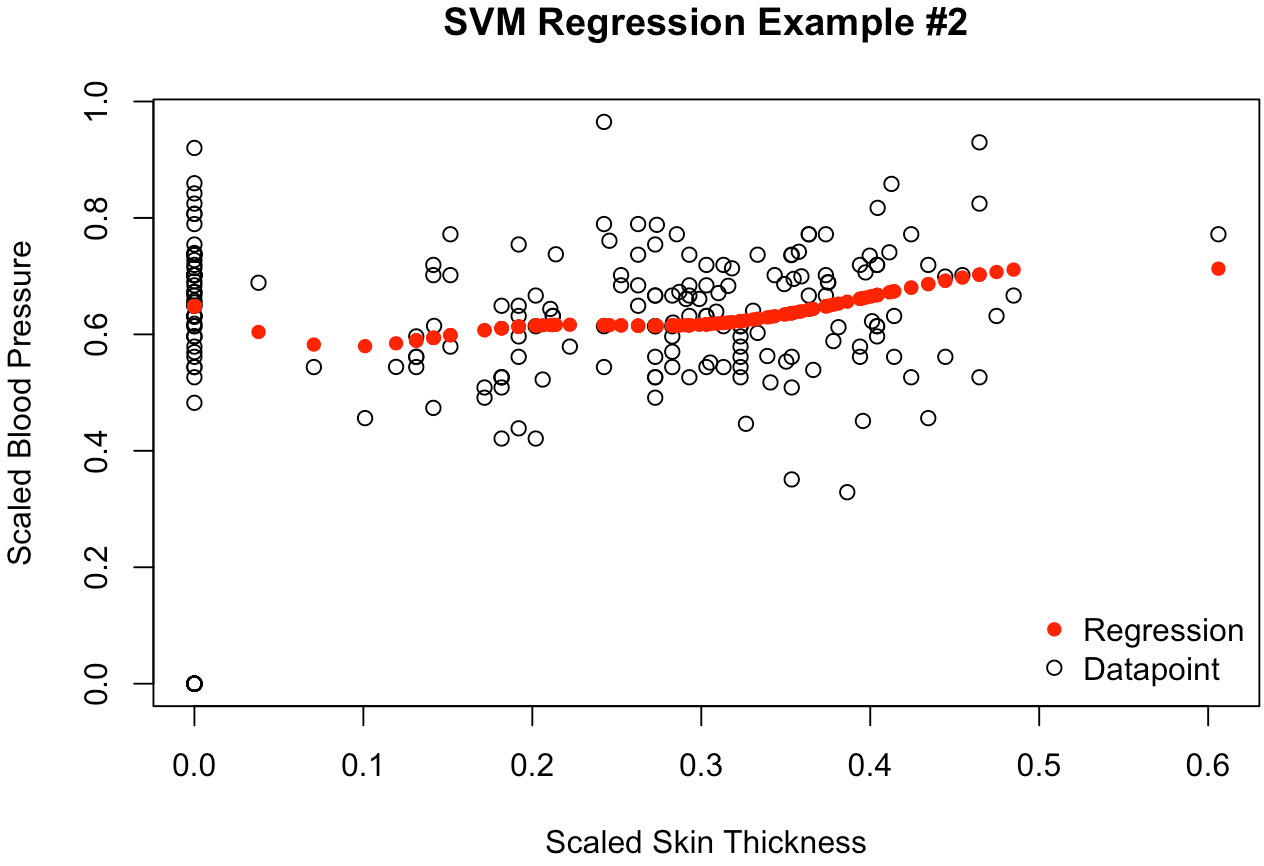
To demonstrate SVMs' usefulness, two regression examples were explored to predict pregnancies and blood pressure. Using the default SVM values, a model was created using the radial kernel the regression line generally follows the data. The root mean square error in both cases was below 17%.

```
# REference: https://www.kdnuggets.com/2017/03/building-regression-models-support-vec
tor-regression.html
modelsvm <- svm(Pregnancies ~ Age, data_train)
predict_pregnancies <- predict(modelsvm, data_test)
p <- plot(x=data_test$Age, y=data_test$Pregnancies, main="SVM Regression Example #1",
          xlab="Scaled Age", ylab="Scaled Pregnancies")
points(data_test$Age, predict_pregnancies, col='red', pch=16)
legend("topright",legend=c("Regression", "Datapoint"),
       col=c("red", "black"), pch = c(16,1), bty="n")
```




```
## [1] "The root mean square error for predicted pregnancies is: 16%."
```

```
modelsvm <- svm(BloodPressure ~ SkinThickness, data_train)
predict_BloodPressure <- predict(modelsvm, data_test)
plot(x=data_test$SkinThickness, y=data_test$BloodPressure,
     main="SVM Regression Example #2",
          xlab="Scaled Skin Thickness", ylab="Scaled Blood Pressure")
points(data_test$SkinThickness, predict_BloodPressure, col='red', pch=16)
legend("bottomright",legend=c("Regression", "Datapoint"),
       col=c("red", "black"), pch = c(16,1), bty="n")
```




```
## [1] "The root mean square error for predicted pregnancies is: 18%."
```

SVM with a Radial Kernel

To explore SVM performance in predicting the outcome variable using the raiaal kernel, we'll be splitting the training data into five folds, and using a grid-search to find the optimal value for the cost (misclassification penalty) parameter.

```
# We'll mutate the data_test & data_train to work with the train function
data_test <- data_test %>% mutate(Outcome = as.factor(ifelse(Outcome == 0, 'No','Yes
')))
data_train <- data_train %>% mutate(Outcome = as.factor(ifelse(Outcome == 0, 'No','Ye
s')))

ctrl <- trainControl(method="repeatedcv",
                     number = 5,                          # 5 folds
                     summaryFunction=twoClassSummary,
                     classProbs=TRUE)

# Grid search to fine tune SVM
grid <- expand.grid(sigma = seq(0.01, 5, 0.2),
                    C = seq(1, 10, by=1)
                    )
```

```
# Tuning is faster in parallel
all_cores <- parallel::detectCores(logical = FALSE)
cl <- makeCluster(all_cores)
registerDoParallel(cl)

svm.tune <- train(x=data_train[,1:8],
                  y= data_train$Outcome,
                  method = "svmRadial",
                  metric="ROC",
                  preProc=c("center"),
                  tuneGrid = grid,
                  trControl=ctrl)
```

After searching for the values of gamma and C (cost parameter), the best values are below:

```
svm.tune$bestTune
```

```
##     sigma C
## 141  2.81 1
```

```
pred <- predict(svm.tune, data_test[,1:8], type="prob")
p <- ifelse(pred$No > 0.5, 'No', 'Yes')
c <- confusionMatrix(data_test$Outcome, as.factor(p))
```

Confusion Matrix - SVM with Radial Kernel

```
c[[2]]
```

```
##          Reference
## Prediction No Yes
##        No  95  17
##        Yes  4  98
```

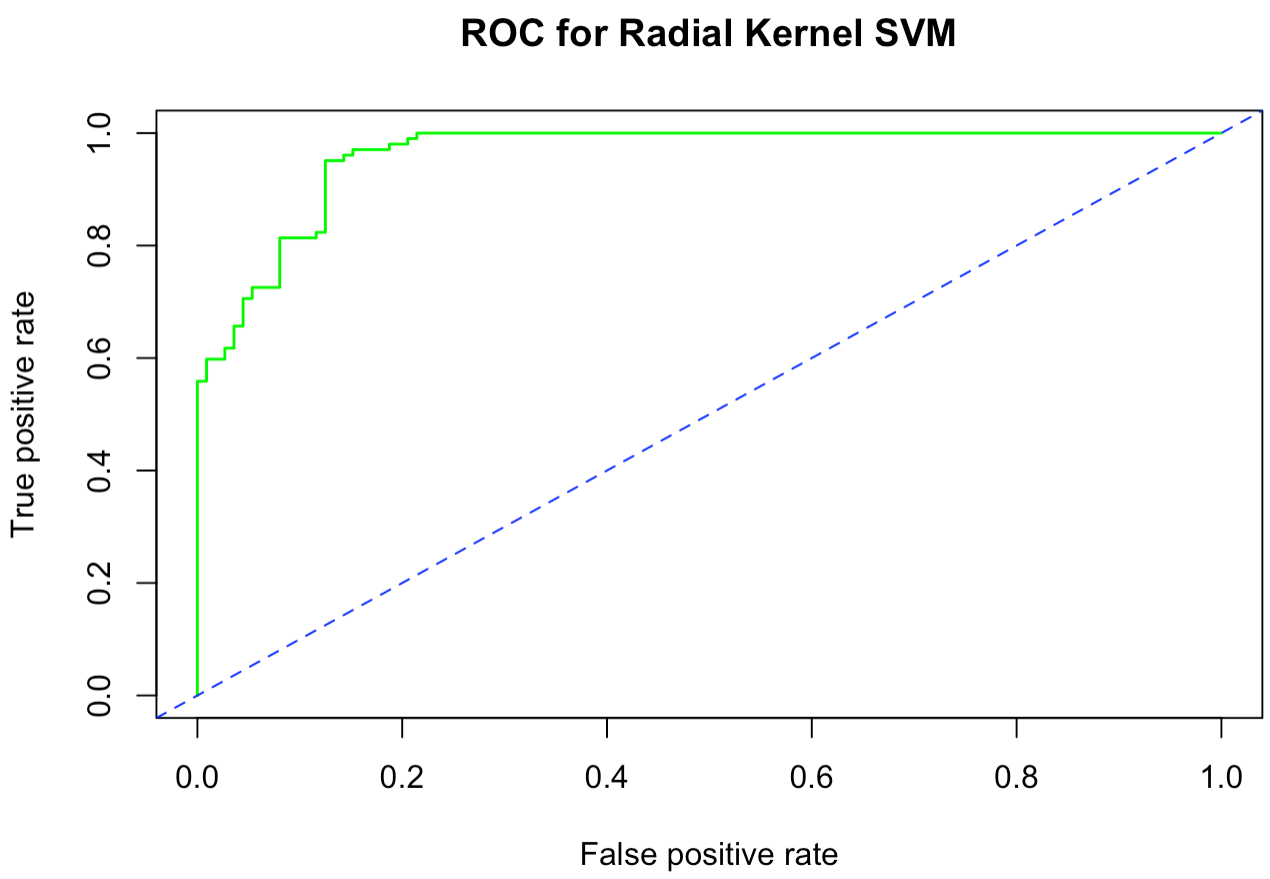
```
radial_accuracy <- c$overall[[1]]

## compare with last assignments RF accuracy
comparison <- ifelse( radial_accuracy > 0.88, 'higher', 'lower')


paste0("SVM with Radial Kernel Accuracy: ", round(radial_accuracy,3)*100, "%")
```

```
## [1] "SVM with Radial Kernel Accuracy: 90.2%"
```

The overall accuracy rate is 90.2%. Notice, this is a litte higher than the RF's accuracy of 88% from the previous assignment. Again, this is the result of tuning the radial kernel (as per recommendation).

This is one other warning from the reference literature: especially in biology or medicine, SVMs may not always perform the best when there is relatively little non-linearity in the data.

From the ROC curve, the accuracy (true postive) rate peaks around 90.2%.



SVM with a Polynomial Kernel

We'll also try to train a SVM model with another popular kernel: polynomial. This model has three hyperparameters, the polynomial degree, scaling and penalty factors.

```
ctrl <- trainControl(method="repeatedcv",
                     number = 5,
                     summaryFunction=twoClassSummary,
                     classProbs=TRUE)

# Grid search to fine tune SVM
grid <- expand.grid(degree = seq(1,5,0.2),
                    scale = c(0.001, 0.01, 0.1, 0.5),
                    C = seq(1, 10, by=1))
```

```
svm.tune <- train(x=data_train[,1:8],
                  y= data_train$Outcome,
                  method = "svmPoly",
                  metric="ROC",
                  preProc=c("center"),
                  tuneGrid = grid,
                  trControl=ctrl)

#svm.tune$results %>% filter(sigma >= 2 & sigma <=3)
```

The best values for this model:

```
svm.tune$bestTune
```

```
##     degree scale C
## 821      5   0.1 1
```

```
pred <- predict(svm.tune, data_test[,1:8], type="prob")
p <- ifelse(pred$No > 0.5, 'No', 'Yes')
c <- confusionMatrix(data_test$Outcome, as.factor(p))
```

Confusion Matrix

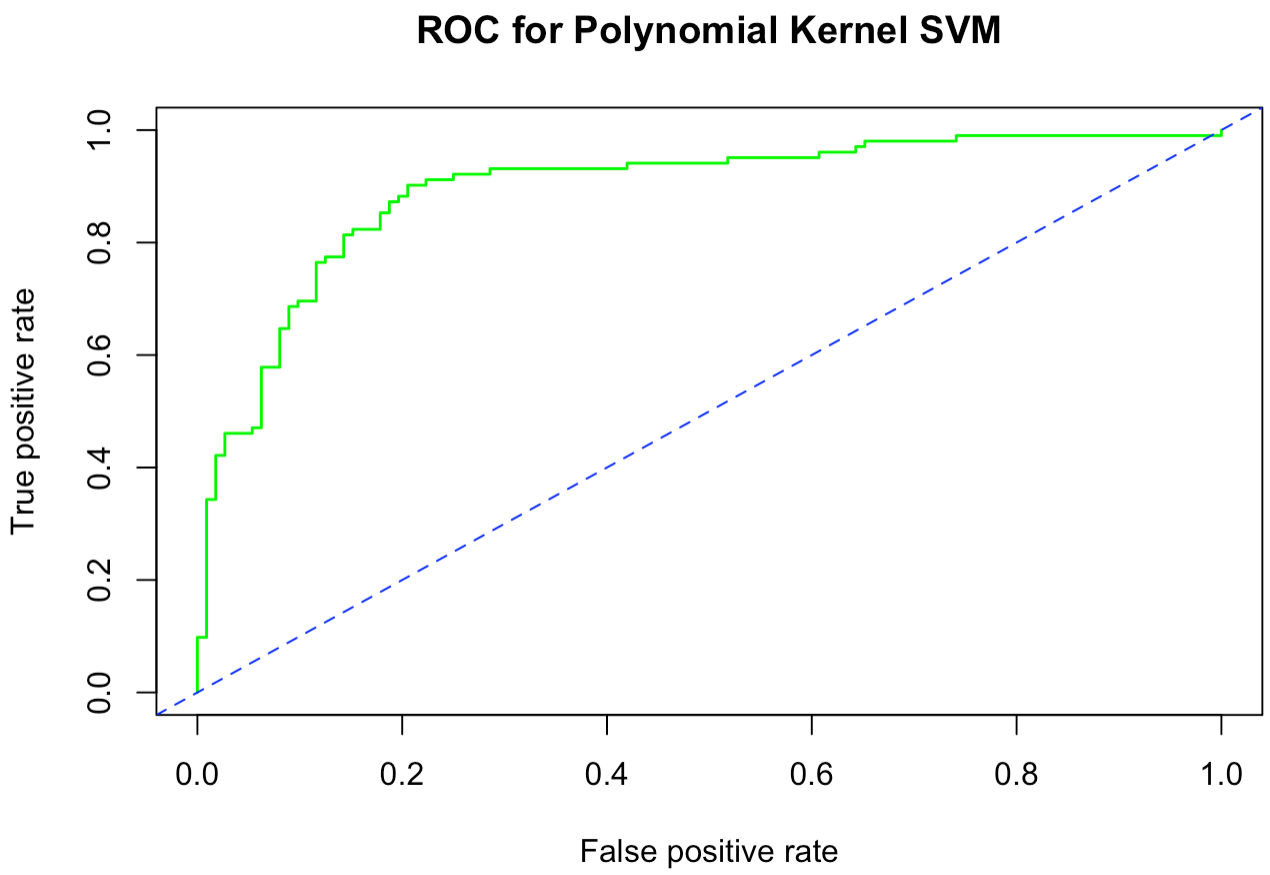
```
c[[2]]
```

```
##           Reference
## Prediction No Yes
##        No  92  20
##        Yes 16  86
```

```
poly_accuracy <- c$overall[[1]]

paste0("SVM with Polynomial Kernel Accuracy: ", round(poly_accuracy,3)*100, "%")
```

```
## [1] "SVM with Polynomial Kernel Accuracy: 83.2%"
```

This model shows a lower accuracy than the SVM with radial kernel.



Conclusion

Again, as other researchers have noted, SVMs are very useful and are frequently the best option for many classification and regression problems, provided you follow their guidelines for data preprocessing and paying attention to the dataset dimensions. However, following these guidelines does not guarentee SVMs will always be the best performing algorithm. In this assignment, we found the tuned SVM model with the radial kernel performs simiarly to RF for the diabetic dataset.

References

1. Zaza, S, Atemkeng, M, Hamlomo, S.:Wine feature importance and quality prediction: A comparative study of machine learning algorithms with unbalanced data (Oct 2023).
2. Akinyemi, M., Yinka-Banjo1, C., Ugot, O.A., Nwachuku, A: Estimating the time-lapse between medical insurance reimbursement with non-parametric regression models.

3. Kadry, S.,Rajinikanth, V.,Rho, S.,Raja, N.S.M. ,Rao, V.S.,Thanaraj, K.P.:Development of a Machine-Learning System to Classify Lung CT Scan Images into Normal/COVID-19 Class
4. Maszczyk, T., Duch, W.: Support Feature Machines: Support Vectors are not enough (2019)
5. Hsu, C.W., Chang, C.C, Lin C.J.:A Practical Guide to Support Vector Classification (2016)