Data 622 - HW 3 - SVM Models

William Jasmine

2024-05-05

0. Setup

The following packages are required to rerun this `.rmd` file:

```
library(tidyverse)
library(rpart)
library(rpart.plot)
library(knitr)
library(gridExtra)
library(ranger)
library(fastDummies)
library(e1071)

seed_num <- 42 # common seed value for randomized operations
set.seed(seed_num)
```

1. Introduction

The purpose of the work presented here is to evaluate the performance resulting from the usage of Support Vector Machine (SVM) algorithms to solve classification problems. Furthermore, the results of these algorithms are compared against various tree based models, specifically those reported in my previously written RPubs article.

The dataset used here was sourced from Kaggle, and contains information on various mushroom species. Information on how the data was originally collected can be found in this article.

2. Import Data

The cell below imports the data from the `mushroom_cleaned.csv` file and stores it as a dataframe, `mush`:

```
mush <- read.csv("data/mushroom_cleaned.csv")
kable(mush[1:5,], caption='First 5 rows of mush dataframe:')
```

Table 1: First 5 rows of mush dataframe:

cap.diameter	cap.shape	gill.attachment	gill.color	stem.height	stem.width	stem.color	season	class
1372	2	2	10	3.807467	1545	11	1.8042727	1
1461	2	2	10	3.807467	1557	11	1.8042727	1
1371	2	2	10	3.612496	1566	11	1.8042727	1

cap.diameter	cap.shape	gill.attachment	gill.color	stem.height	stem.width	stem.color	season	class
1261	6	2	10	3.787572	1566	11	1.8042727	1
1305	6	2	10	3.711971	1464	11	0.9431946	1

The rest of this section is copied directly from Section 2 of the aforementioned RPubs article:

Each row in the `mush` dataframe represents measurements taken for a single mushroom, with the `class` field in `mush` referring to whether or not the mushroom is edible. This is a binary field with each value representing the following:

- 0 = edible/non-poisonous
- 1 = non-edible/poisonous

The remaining eight fields are all feature variables that quantitatively/qualitatively describe the mushroom, and are defined as follows:

Field Name	Variable Type/Data Type	Description
`cap.diameter`	Continuous/Float	The diameter of the mushroom's cap in m^{-4}.
`cap.shape`	Categorical/Integer	A value from 0-6 the represents the shape of the mushroom cap. These include "bell", "conical", "convex", "flat", "sunken", "spherical", and "others", respectively.
`gill.attachment`	Categorical/Integer	A value from 0-6 the represents how the mushroom gills are connected to its cap. These include "adnate", "adnexed", "decurrent", "free", "sinuate", "pores", and "unknown", respectively.
`gill.color`	Categorical/Integer	A value from 0-11 the represents the color of the mushroom's gills. These include "brown", "buff", "gray", "green", "pink", "purple", "red", "white", "yellow", "blue", "orange", and "black", respectively.
`stem.height`	Continuous/Float	The height of the mushroom's stem (base to cap) in cm.
`stem.width`	Continuous/Float	The width of the mushroom's stem (at it's base) in $^{-4}$.
`stem.color`	Categorical/Integer	A value from 0-11 the represents the color of the mushroom's cap. See `gill.color` for list of colors.
`season`	Continuous/Float	A value that represents the time of year the measurements for the mushroom were taken. Larger values represent measurements that were taken further along in the year.

Based on the below, the `mush` dataframe contains measurements of 53,035 mushrooms:

```
nrow(mush)
```

```
## [1] 54035
```

Of these observations, the following output summarizes the quantity of each that were classified as edible/non-edible:

```
table(mush$class)
```

```
##
##     0     1
## 24360 29675
```

3. Data Cleaning Steps

SVM models, unlike their tree based counterparts, do not do well with large amounts of data and can can be difficult to train with too many observations. As such, the cell below randomly samples 10,000 observations from `mush` (reducing it's size by more than 75%):

```
set.seed(seed_num)

n <- 10000
sample_choices <- sample(1:nrow(mush), n, replace=FALSE)
mush <- mush[sample_choices,]
```

The cell below replicates the rest of the data cleaning steps carried out in Section 3 of the aforementioned RPubs article.

```
# standardize length measurements
mush$cap.diameter <- mush$cap.diameter / 10
mush$stem.height <- mush$stem.height * 10
mush$stem.width <- mush$stem.width / 10


# balance classes by undersampling
set.seed(seed_num)
class_0 <- which(mush$class == 0)
class_1 <- which(mush$class == 1)
class_1 <- sample(class_1, size=length(class_0), replace=FALSE)
selected_rows <- c(class_0, class_1)
mush <- mush[selected_rows,]

# cast class field as a factor
mush$class <- as.factor(mush$class)
```

Section 4 - SVM Models

Section 4.1 - Prepare Data

Before building and evaluating any SVM models, we need to split the data in `mush` into training and testing sets that can be used to properly evaluate their performance:

```r
set.seed(seed_num)

# define training size
train_size <- 0.75

# create separate dataframes by class
class_0_df <- mush[which(mush$class == 0),]
class_1_df <- mush[which(mush$class == 1),]

# determine row numbers of samples
n <- floor(nrow(class_0_df) * train_size)
sample_vec <- 1:nrow(class_0_df)
train_samples <- sample(sample_vec, size=n, replace=FALSE)

# sample the class dataframes
train_0_df <- class_0_df[train_samples,]
test_0_df <- class_0_df[-train_samples,]

train_1_df <- class_1_df[train_samples,]
test_1_df <- class_1_df[-train_samples,]

# recombine dataframes
train <- rbind(train_0_df, train_1_df)
test <- rbind(test_0_df, test_1_df)

# shuffle rows
train <- train[sample(1:nrow(train)), ]
test <- test[sample(1:nrow(test)), ]

# split into x and y
train_x <- select(train, -class)
train_y <- select(train, class)

test_x <- select(test, -class)
test_y <- select(test, class)
```

Section 4.2 - SVM Model With Two Features

Simply put, SVM models determine what's known as a "decision boundary" in n-dimensional space, such that n is the number of explanatory fields. Predictions can be made using these models by determining where new observations are located in this space in reference to the decision boundary.

Because this decision boundary can only be easily visualized in two or three dimensions, we will first build a model using just two explanatory fields. While this model is not expected to have exceedingly high accuracy, it will provide insight into how a SVM model makes predictions. The two features chosen in this case are those deemed to be the two "most important" continuous features of the random forest built in Section 6 of the previous RPubs article: `stem.width` and `cap.diameter`.

```r
set.seed(seed_num)

svm_data <- train[c('stem.width', 'cap.diameter', "class")]
simple_svm <- svm(class~.,data=svm_data, kernel="linear")
print(simple_svm)
```

```
##
## Call:
## svm(formula = class ~ ., data = svm_data, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  5874
```

Next, we can use the model to make predictions on the test set:

```
preds <-unname(predict(simple_svm, test[c('stem.width', 'cap.diameter')]))
confusion_matrix <- table(preds, test$class)
print(confusion_matrix)
```

```
##
## preds   0   1
##     0 609 416
##     1 520 713
```

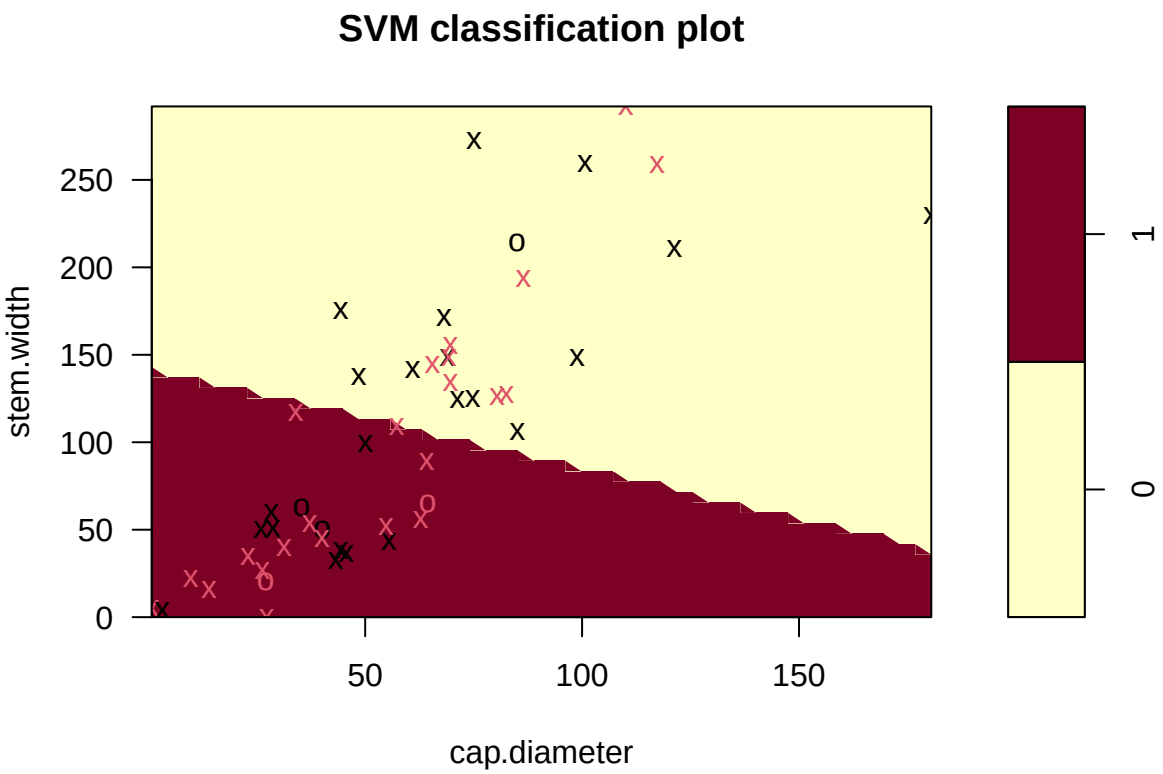
As we can see in the above confusion matrix, the model did not perform very well: it only correctly predicted 609 of the 1,126 edible mushrooms (~51%) and 713 of the 1,126 poisonous mushrooms (~63%) resulting in a total accuracy score of:

```
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Accuracy:", round(accuracy, 2)))
```

```
## [1] "Accuracy: 0.59"
```

While this result evidences a lack of promising utility for this model (it only performs 9% better than chance), we can see clearly how it is making its decisions:

```
set.seed(seed_num)
s_rows <- sample(1:nrow(test), 50, replace=FALSE)
s <- test[s_rows,c('stem.width', 'cap.diameter', 'class')]
plot(simple_svm, data = s)
```



In the above plot we can see the linear decision boundary drawn by the SVM model, in which an observation that places below the boundary is classified as belonging to class 0 (edible). Based on the decision boundary, mushrooms with larger stem widths and cap diameters are more likely to be edible, with the opposite being true for poisonous mushrooms. The points on the graph represent predictions made using 50 randomly chosen observations from the test set, which make it clear that this model is under-performing: red X's in the top section and black X's in the bottom section indicate incorrect predictions.

Section 4.3 - Using All Features

While the previously created SVM model was useful in visualizing how SVM models operate, it was not an effective classifier. Using only two features resulted in a decision boundary that was not complex enough to capture many of the underlying connections/intricacies of our dataset. As such, building a more complex model should provide better results. The cell below creates a new SVM model using all the explanatory fields in `mush`:

```
set.seed(seed_num)

complex_svm <- svm(class~.,data=train)
print(complex_svm)
```

```
##
## Call:
## svm(formula = class ~ ., data = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
```

```
##
## Number of Support Vectors:  3570
```

We can now use this model to make predictions on the test set:

```
preds <-unname(predict(complex_svm, test))
confusion_matrix <- table(preds, test$class)
print(confusion_matrix)
```

```
##
## preds   0   1
##     0 964 209
##     1 165 920
```

The results above show a massive improvement compared to the previously created two-feature model: it correctly predicted 1,076 of the 1,126 edible mushrooms (~95%) and 1,022 of the 1,126 poisonous mushrooms (~91%) resulting in a total accuracy score of:

```
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Accuracy:", round(accuracy, 2)))
```

```
## [1] "Accuracy: 0.83"
```

It is clear that this SVM model produces high-quality predictions, and its success is starting to rival that of the best tree based model produced here (the random forest from section 6, which had an accuracy score of almost 99%).

Section 4.3 - Tuning the Model

To see if a SVM model can perform as well as the random forest from the previous analysis, we can attempt to tune the hyperparameters that the SVM model takes as inputs.

All SVM models require specifying a "kernel", which dictates the functions used to determine the support vectors that draw the decision boundary. One of the most common kernels chosen is known as the "radial basis function" (RPS), which is controlled by a hyperparameter, γ. In short, γ dictates how much influence a single training example has over the decision boundary, with higher values resulting in more complex boundaries.

The Cost C is another (kernel-independent) hyperparameter that controls how much the model should care about an incorrectly classified observation. Like models with large γ, models with high C tend to have more complex decision boundaries.

The cell below uses the RPS kernel with four different values of these two hyperparameters to determine the values of each that had the best results:

```
gammas <- 10^(-1:2)
costs <- 10^(-1:2)

tuned_model_rps <-
  tune.svm(class~., data = train, kernel="radial",
           cost = costs, gamma = gammas)
tuned_model_rps$best.parameters
```

```
##    gamma cost
## 10     1   10
```

Of $4 \cdot 4 = 16$ different SVM models were tested, the one using $\gamma = 0.1$ and $C = 100$ was found to produce the best results. We can now input these parameters into a new model to make predictions on the test set:

```r
tuned_svm <- svm(class~.,data=train, kernel='radial', cost=100, gamma=0.1)
preds <-unname(predict(complex_svm, test))
confusion_matrix <- table(preds, test$class)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Accuracy:", round(accuracy, 2)))
```

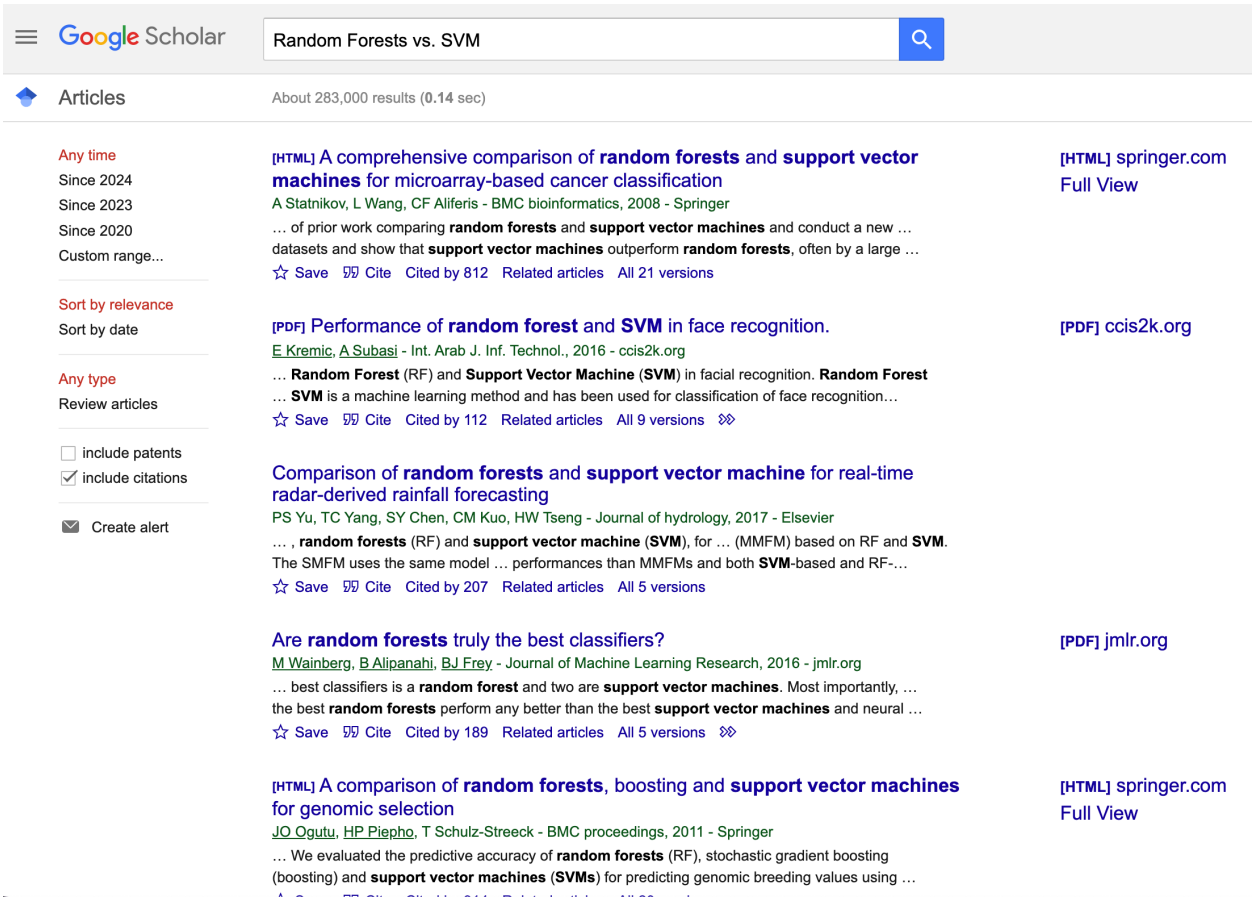
```
## [1] "Accuracy: 0.83"
```

Despite the search, the model using these parameters performed just as well as the "out-of-the-box" SVM model.

Section 5 - Conclusion

As the work presented here has shown, SVM models can be quite an effective tool for solving classification problems and have the ability to achieve high levels of accuracy. However, for the use case presented here, the tree-based models produced previously yielded better results. Not only this, but the previously created random forest was able to ingest and train all observations in an extremely short period of time. With the SVM models, training times on the original `mush` dataframe took much too long to be feasible, resulting in number of observations having to be reduced by almost 75%. Even after this reduction, training times were still an issue with hyperparameter tuning: though 16 different SVM input-sets were tested, it did not result in a model that exceeded the quality of the "out-of-the-box" predictions. This is not to say that there aren't improvements that can be made to the model, but they would take a not insignificant amount of time to determine. Some general guidelines based off these results seem to indicate that random forests work best for large datasets that contain a multitude of categorical and continuous fields, while SVMs work best on smaller datasets containing complex relationships between continuous fields.

However, it seems as though the main point of the work presented both here and here is that the model chosen should be evaluated depending on the use case. In this instance it appears as though random forests are the best choice, but this is most definitely not ubiquitous for all classification problems. Clear evidence for this can be seen by performing a literature search of how random forests compare against SVMs:



It clear from the above that the question of random forests vs SVM models has been evaluated for numerous classification problems across a plethora of academic fields.

To conclude, both tree-based and SVM models can be top choices when attempting to build superior classifier, but it is the responsibility of the data practitioner building these models to determine what is most pertinent to their use case.

Section 6 - Sources

The work presented here incorporates knowledge obtained from the following sources.

1. Tsang, I., Kwok J., Cheung P. , "Core Vector Machines, Fast SVM Training on Very Large Data Sets", Journal of Machine Learning Research, vol. 6, April 2005, pp. 363-392 - Helpful to understanding how dataset size impacts training time and performance of SVM models.
2. Guido, R., Carmela-Groccia, M., Conforti, D., "A Hyper-Parameter Tuning Approach for Cost-Sensitive Support Vector Machine Classifiers", Soft Computing, vol. 27, February 2022, pp. 12863-12881 - Helpful for understanding how different hyperparameters affect the SVM model.
3. Statnikov, A. & Aliferis, C., "Are Random Forests Better Than Support Vector Machines for Microarray-Based Cancer Reseach", AMIA Annu Symp Proc, March 2007, pp. 686-690 - Helpful for understanding when to use random forests vs. SVM models.