Support Vector Machines

Brooke Anderson

February 9, 2016

```r
library(dplyr) # Data wrangling
library(tidyr) # Data wrangling
library(caret) # Machine learning
library(ggplot2) # Plotting
library(VIM) # k-NN imputation
library(stringr) # For string manipulation
```

```r
train <- read.csv("data/train.csv") %>%
         select(Survived, Pclass, Sex, Age) %>%
         mutate(Survived = factor(Survived),
         Pclass = ordered(Pclass),
         Sex = factor(Sex))
test <- read.csv("data/test.csv") %>%
        select(Pclass, Sex, Age) %>%
        mutate(Pclass = ordered(Pclass),
        Sex = factor(Sex))
test_ids <- read.csv("data/test.csv") %>%
            select(PassengerId)
```

The `caret` webpage has a great list of all the different models that you can fit using `method`, including the tuning parameters for each.

Fitting an SVM using `train`

```r
set.seed(1201)
svm_fit <- train(Survived ~ .,
                data = train,
                method = "svmRadial",
                preProc = c("center", "scale"),
                tuneLength = 10,
                trControl = trainControl(method = "cv"))
svm_fit
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 891 samples
##   3 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered, scaled
## Resampling: Cross-Validated (10 fold)
##
## Summary of sample sizes: 642, 643, 642, 643, 643, 643, ...
##
```

```
## Resampling results across tuning parameters:
##
##   C       Accuracy   Kappa      Accuracy SD  Kappa SD
##     0.25  0.7858372  0.5326963  0.04201414   0.10435715
##     0.50  0.7815923  0.5271587  0.04079005   0.10050388
##     1.00  0.7815532  0.5290966  0.03940454   0.09439006
##     2.00  0.7801056  0.5244846  0.04671379   0.10874425
##     4.00  0.7787167  0.5213073  0.04465780   0.10428684
##     8.00  0.7815728  0.5279817  0.04300597   0.10138178
##    16.00  0.7829617  0.5307300  0.04360245   0.10241929
##    32.00  0.7787559  0.5219467  0.03740498   0.09049189
##    64.00  0.7802034  0.5265803  0.04749536   0.11487567
##   128.00  0.7830008  0.5328598  0.04713419   0.11334024
##
## Tuning parameter 'sigma' was held constant at a value of 1.010875
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were sigma = 1.010875 and C = 0.25.
```

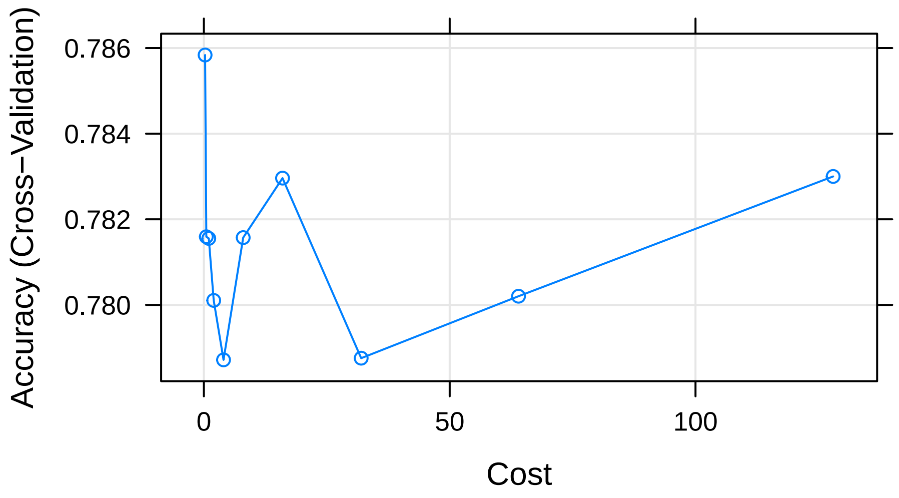
Note: Running this required me to install a new package (`kernlab`), which I evidently didn't have installed yet. I got the help for this new package using `??"kernlab"`.

If I used a bootstrap method, I got the following warning message when I ran the above code (a bunch of them):

```
Warning message:
In data.row.names(row.names, rowsi, i) :
  some row.names duplicated: ...
```

I don't think this should be a concern– I think it's just where the training sometimes resamples the same rows if you do something like bootstrapping.

```r
plot(svm_fit)
```



This training is only tuning C, not σ. From the JSS article on `caret`, explaining this (Kuhn 2008):

> "For this particular model, it turns out that there is an analytical method for directly estimating a suitable value of σ from the training data (Caputo et al. 2002). By default, the train function uses the sigest function in the kernlab package (Karatzoglou et al. 2004) to initialize this parameter. In doing this, the value of the cost parameter C is the only tuning parameter."

Regarding the tuning length for SVM, from the same paper:

> "tuneLength: controls the size of the default grid of tuning parameters. For each model, train will select a grid of complexity parameters as candidate values. For the SVM model, the function will tune over C = 10^{-1}; 1; 10. To expand the size of the default list, the tuneLength argument can be used. By selecting tuneLength = 5, values of C ranging from 0.1 to 1,000 are evaluated."

```
preds <- predict(svm_fit, newdata = test)
```

This still leaves us without predictions for our missing values– it only predicts for complete cases in the testing data:

```
length(preds)
```

```
## [1] 332
```

```
dim(test)
```

```
## [1] 418   3
```

```
sum(complete.cases(test))
```

```
## [1] 332
```

This time, to handle the missing values, I tried imputing the `test` data using k-NN (with Gower for categorical variables) to "fill in" missing values before I tried to predict all the observations.

```
test2 <- kNN(test)
```

```
## Time difference of 0.01923203 secs
```

```
head(test2)
```

```
##   Pclass    Sex  Age Pclass_imp Sex_imp Age_imp
## 1      3   male 34.5      FALSE   FALSE   FALSE
## 2      3 female 47.0      FALSE   FALSE   FALSE
## 3      2   male 62.0      FALSE   FALSE   FALSE
## 4      3   male 27.0      FALSE   FALSE   FALSE
## 5      3 female 22.0      FALSE   FALSE   FALSE
## 6      3   male 14.0      FALSE   FALSE   FALSE
```

Notice that this imputation adds some columns to let you know which variables were imputed for each observation.

```
preds2 <- predict(svm_fit, newdata = test2)
length(preds2)
```

```
## [1] 418
```

Now I'll write that out and test it on Kaggle…

```
out <- data.frame(PassengerId = test_ids,
                  Survived = as.numeric(as.character(preds2)))
write.csv(out, file = "predictions/SVM_Pclass_Sex_Age_imputed.csv",
          row.names = FALSE)
```

The Kaggle score was 0.75598.

For a comparison, I also submitted a set of predictions where I always used 0 as the prediction for observations with one or more missing predictors:

```
out2 <- data.frame(PassengerId = test_ids,
                  Survived = 0)
out2[complete.cases(test), "Survived"] <- as.numeric(as.character(preds))
write.csv(out2, file = "predictions/SVM_Pclass_Sex_Age_all0.csv",
          row.names = FALSE)
```

The Kaggle score for these predictions was 0.75598, which is exactly the same as when imputing data.

If I compare the results of the two predictions, there are only three test observations that have different answers for the two methods of handling missing values:

```
sum(out != out2)
```

```
## [1] 3
```

This is out of 86 observations in the testing dataset with missing predictors, so the two methods of handling test observations with missing data seem to agree pretty well (at least in this case and for these predictive variables).

Including more predictors

Next, I'll try to include more of the predictive variables:

```
train <- read.csv("data/train.csv") %>%
         select(Survived, Pclass, Name, Sex, Age, SibSp, Parch, Fare, Embarked) %>%
         mutate(Survived = factor(Survived),
         Pclass = ordered(Pclass),
         Name = gsub("[\\,\\.\\ ]", "", str_extract(Name, ",\\ .+?\\.")),
         Name = factor(ifelse(Name %in% c("Mr", "Mrs", "Miss", "Master"),
                              Name, "Other")),
         Sex = factor(Sex),
         Embarked = factor(ifelse(Embarked == "", NA, as.character(Embarked))))
test <- read.csv("data/test.csv") %>%
         select(Pclass, Name, Sex, Age, SibSp, Parch, Fare, Embarked) %>%
         mutate(Pclass = ordered(Pclass),
         Name = gsub("[\\,\\.\\ ]", "", str_extract(Name, ",\\ .+?\\.")),
         Name = factor(ifelse(Name %in% c("Mr", "Mrs", "Miss", "Master"),
                              Name, "Other")),
         Sex = factor(Sex),
         Embarked = factor(ifelse(Embarked == "", NA, as.character(Embarked))))
test_ids <- read.csv("data/test.csv") %>%
            select(PassengerId)
```

```r
set.seed(1201)
svm_fit <- train(Survived ~ .,
                data = train,
                method = "svmRadial",
                preProc = c("center", "scale"),
                tuneLength = 10,
                trControl = trainControl(method = "cv"))
svm_fit
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 891 samples
##   8 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered, scaled
## Resampling: Cross-Validated (10 fold)
##
## Summary of sample sizes: 640, 641, 640, 642, 641, 641, ...
##
## Resampling results across tuning parameters:
##
##   C       Accuracy   Kappa      Accuracy SD  Kappa SD
##     0.25  0.8272535  0.6338414  0.04414908   0.09848743
##     0.50  0.8230869  0.6224617  0.04911034   0.11006873
##     1.00  0.8244757  0.6233879  0.04956471   0.11058077
##     2.00  0.8272926  0.6286976  0.05119419   0.11372670
##     4.00  0.8160049  0.6048916  0.03989100   0.09127420
##     8.00  0.7963839  0.5640132  0.04584743   0.10461894
##    16.00  0.7865627  0.5431288  0.04496733   0.10141082
##    32.00  0.7839040  0.5375092  0.04804285   0.10991665
##    64.00  0.7782696  0.5264045  0.04635028   0.10652651
##   128.00  0.7811066  0.5340010  0.05089721   0.11496562
##
## Tuning parameter 'sigma' was held constant at a value of 0.09284051
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were sigma = 0.09284051 and C = 2.
```

This time, to handle the missing values, I tried imputing the `test` data using k-NN (with Gower for categorical variables) to “fill in” missing values before I tried to predict all the observations.

```r
test2 <- kNN(test)
```

```
## Time difference of 0.03684497 secs
```

```r
head(test2)
```

```
##   Pclass Name    Sex  Age SibSp Parch    Fare Embarked Pclass_imp Name_imp
## 1      3   Mr   male 34.5     0     0  7.8292        Q      FALSE    FALSE
## 2      3  Mrs female 47.0     1     0  7.0000        S      FALSE    FALSE
## 3      2   Mr   male 62.0     0     0  9.6875        Q      FALSE    FALSE
## 4      3   Mr   male 27.0     0     0  8.6625        S      FALSE    FALSE
## 5      3  Mrs female 22.0     1     1 12.2875        S      FALSE    FALSE
## 6      3   Mr   male 14.0     0     0  9.2250        S      FALSE    FALSE
##   Sex_imp Age_imp SibSp_imp Parch_imp Fare_imp Embarked_imp
## 1   FALSE   FALSE     FALSE     FALSE    FALSE        FALSE
## 2   FALSE   FALSE     FALSE     FALSE    FALSE        FALSE
## 3   FALSE   FALSE     FALSE     FALSE    FALSE        FALSE
## 4   FALSE   FALSE     FALSE     FALSE    FALSE        FALSE
## 5   FALSE   FALSE     FALSE     FALSE    FALSE        FALSE
## 6   FALSE   FALSE     FALSE     FALSE    FALSE        FALSE
```

Notice that this imputation adds some columns to let you know which variables were imputed for each observation.

```
preds2 <- predict(svm_fit, newdata = test2)
length(preds2)
```

```
## [1] 418
```

Now I'll write that out and test it on Kaggle…

```
out <- data.frame(PassengerId = test_ids,
                  Survived = as.numeric(as.character(preds2)))
write.csv(out, file = "predictions/SVM_kitchen_sink_imputed.csv",
          row.names = FALSE)
```

The Kaggle score was 0.78469, which is the best I've gotten so far (but probably still not as good as some tree ensembles that include title, based on the forums / tutorials).

Some interesting comments from the forums

Regarding what a "good" score would be:

> "82% would be a very good score. 84% would be an amazing score. If you're getting 84 then I think there's very little left that you can learn from plugging away any further on this challenge and you should move on to new challenges. Considering that most people above 0.85 are almost definitely cheating (it's a"toy" competition with no prize), a score of 0.82 would put you in roughly the top 0.5% of submissions – a great result. Your profile says you've got 0.785 at the moment, so there's some room for improvement. Also, keep in mind that the public ranking is not going to be the final ranking."

> "The problem with predicting NA's is, that you also add noise in your inputs. If it is still worth it, depends very much on the data. In this challenge i remember that using name features could be used to guess missing age values (Master for young boys etc.). This is a good way, as the name is otherwise hard to extract information out. By using the name for predicting the NA's you add information so to say. This is the way to go for increasing the accuracy: Try to extract features which contain information. If youre able to enrichen your model with more information you might get very good results in this. It is also partly possible to see family relationships through the names, which can be again used for increasing the accuracy. Scores till 85% and probably 90% are possible without any cheating."