Performance Measures

Taylor Shrode

November 1, 2020

Introduction

Accuracy is an important performance measure, but it is wise to collect additional information (Lantz, 2015). "For example, suppose that for 99,990 out of 100,000 newborn babies, a classifier correctly predicted whether the newborn was a carrier of a treatable but potential fatal genetic defect (Lantz, 2015)." This implies the classifier has an accuracy of 99.99% and a 0.01% error rate. However, suppose the genetic defect is found in only 10 out of every 100,000 babies. If the classifier predicts *no defect* for all babies, then the classifier is correct for 99.99 percent of all cases, but incorrect for 100% of the cases with the defect (Lantz, 2015). This is one reason that other performance measures need to be analyzed.

For this assignment, we will be exploring various performance measures including Cohen's kappa, precision, recall, specificity, and the F1 measure. These are used to test the validity and reliability of a model/measure. Validity is the accuracy of a measure and reliability is used to measure the consistency of a measure (WorldClass FTE). The data set we will be using contains values for six biomechanical features. These features are used to classify orthopedic patients into two classes (Normal/Abnormal). Each patient (observation) is represented in the data set by six biomechanical attributes derived from the shape and orientation of the pelvis and lumbar spine (Vertebral column dataset, n.d.). These attributes include (Vertebral column dataset, n.d.):

1. Pelvic Incidence
2. Pelvic Tilt
3. Lumbar Lordosis Angle
4. Sacral Slope
5. Pelvic Radius
6. Grade of Spondylolisthesis
7. Class (Normal/Abnormal)

It should also be noted that according to the data set description, the data set does not contain any missing values. To explore the performance measures mentioned above, we will use a supervised learning classifier, Support Vector Machine, to classify our data into classes and then analyze the performance of our model.

Load Libraries and Data Set

Before we build our classifier, we need to load the necessary packages into R.

```
library(DataExplorer)
library("vcd")
library(caret)
library(kernlab)
library(dplyr)
library(gmodels)
```

The **DataExplorer** library allows us to perform data exploration analysis. The **vcd** package is loaded to calculate the kappa statistic. The **caret** package is loaded to partition our data into training and testing data sets, as well as calculating precision, recall, specificity, and create a confusion matrix. The **kernlab** package is loaded so that we can build our SVM classifier. The **dplyr** package is loaded so we can scale our numeric attributes. Finally, the **gmodels** packages is loaded so we can create a two-way cross-tabulation (**CrossTable()**) to examine the relationship between our two nominal variables (Lantz, 2015).

Now, we can load our data into R.

```
column_df <- read.csv(file.choose(), header = TRUE)
head(column_df)
```

```
##   pelvic_incidence pelvic_tilt lumbar_lordosis_angle sacral_slope
pelvic_radius
## 1         63.02782   22.552586              39.60912     40.47523
98.67292
## 2         39.05695   10.060991              25.01538     28.99596
114.40542
## 3         68.83202   22.218482              50.09219     46.61354
105.98513
## 4         69.29701   24.652878              44.31124     44.64413
101.86849
## 5         49.71286    9.652075              28.31741     40.06078
108.16872
## 6         40.25020   13.921907              25.12495     26.32829
130.32787
##   degree_spondylolisthesis    class
## 1                -0.254400 Abnormal
## 2                 4.564259 Abnormal
## 3                -3.530317 Abnormal
## 4                11.211523 Abnormal
## 5                 7.918501 Abnormal
## 6                 2.230652 Abnormal
```

Now, we will look at the structure and a summary of our data set.

```
str(column_df)
```

```
## 'data.frame':    310 obs. of  7 variables:
##  $ pelvic_incidence        : num  63 39.1 68.8 69.3 49.7 ...
##  $ pelvic_tilt             : num  22.55 10.06 22.22 24.65 9.65 ...
##  $ lumbar_lordosis_angle   : num  39.6 25 50.1 44.3 28.3 ...
##  $ sacral_slope            : num  40.5 29 46.6 44.6 40.1 ...
```

```
##  $ pelvic_radius           : num  98.7 114.4 106 101.9 108.2 ...
##  $ degree_spondylolisthesis: num  -0.254 4.564 -3.53 11.212 7.919 ...
##  $ class                   : chr  "Abnormal" "Abnormal" "Abnormal"
"Abnormal" ...
```

```
summary(column_df)
```

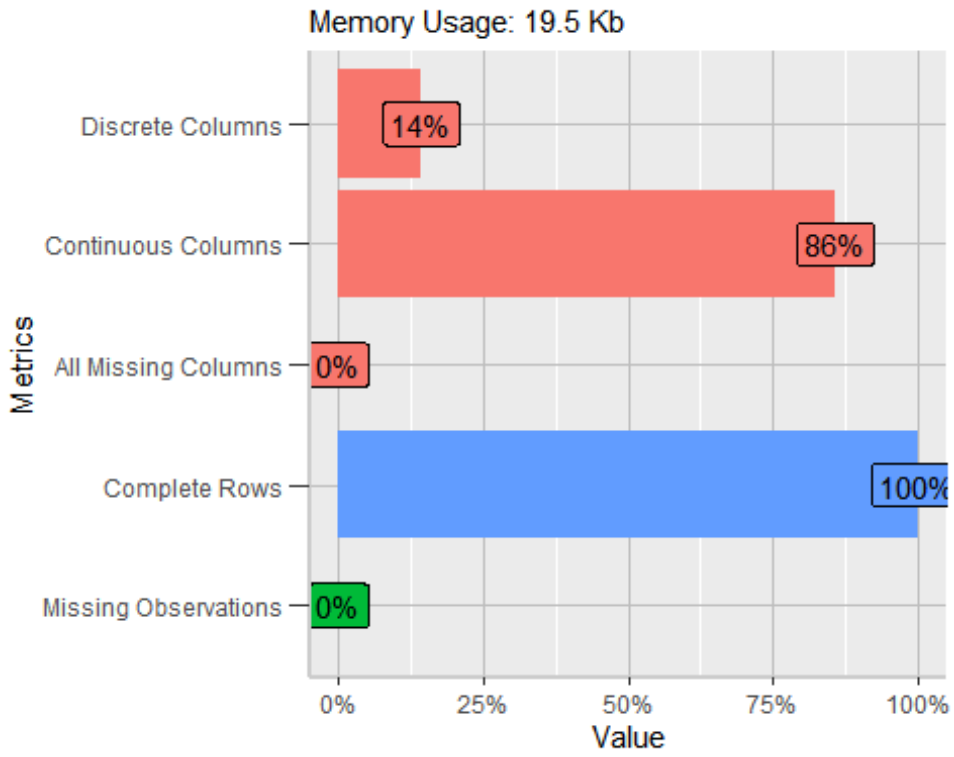
```
##  pelvic_incidence  pelvic_tilt     lumbar_lordosis_angle  sacral_slope
##  Min.   : 26.15   Min.   :-6.555   Min.   : 14.00        Min.   : 13.37
##  1st Qu.: 46.43   1st Qu.:10.667   1st Qu.: 37.00        1st Qu.: 33.35
##  Median : 58.69   Median :16.358   Median : 49.56        Median : 42.40
##  Mean   : 60.50   Mean   :17.543   Mean   : 51.93        Mean   : 42.95
##  3rd Qu.: 72.88   3rd Qu.:22.120   3rd Qu.: 63.00        3rd Qu.: 52.70
##  Max.   :129.83   Max.   :49.432   Max.   :125.74        Max.   :121.43
##  pelvic_radius    degree_spondylolisthesis    class
##  Min.   : 70.08   Min.   :-11.058          Length:310
##  1st Qu.:110.71   1st Qu.:  1.604          Class :character
##  Median :118.27   Median : 11.768          Mode  :character
##  Mean   :117.92   Mean   : 26.297
##  3rd Qu.:125.47   3rd Qu.: 41.287
##  Max.   :163.07   Max.   :418.543
```

From the **str()** output, we notice that our data set contains 310 observations with 7 attributes. We also see that our **class** attribute is considered a character (*chr*) rather than a factor, which we will convert later. From our **summary()** output, we see that our numeric attributes have wide ranges of values. To avoid having any dominating features, we will need to normalize, or scale, these columns.

```
introduce(column_df)
```

```
##   rows columns discrete_columns continuous_columns all_missing_columns
## 1  310       7                1                   6                   0
##   total_missing_values complete_rows total_observations memory_usage
## 1                    0           310               2170        19952
```

```
plot_intro(column_df, theme_config = list(plot.title = element_text(size =
rel(1)), plot.background = element_rect(fill = "white"), panel.grid.major =
element_line(colour = "grey"), axis.line = element_line(size = 1, colour =
"grey80"), axis.ticks.length.y = unit(.25, "cm"), legend.position = "none"))
```



The output above confirms that there are no missing observations in our data set.

Prepare Data

As mentioned above, we will be using Support Vector Machines to classify our data and then apply various performance measures to our model. Support Vector Machines aim to create a hyperplane, which is a flat boundary that divides the space to create homogeneous partitions on either side (Lantz, 2015). SVMs are mostly used for binary classifications, but can be applied to multi-class classification problems. We will be using the our SVM for binary classification and we will utilize the linear kernel. Before, we build our SVM classifier, we need to normalize, or scale, our data and convert our response variable to a factor

From the summary output above, we can see that our observations contain postive and negative values, so we will scale our data instead of applying a normalization function to our data. The **scale()** function "is a generic function whose default method centers and/or scales the columns of a numeric matrix (Scale function, n.d.)." To leave our attributes as numeric values, we use the **mutate_if()** function (Scaling numeric values only in a dataframe with mixed types, n.d.).

```
column_scale <- column_df[1:6] %>% mutate_if(is.numeric, scale) #scale
numeric values
head(column_scale)
```

```
##   pelvic_incidence pelvic_tilt lumbar_lordosis_angle sacral_slope
pelvic_radius
```

```
## 1        0.1468489   0.5005594            -0.6641032   -0.1846517    -
1.4453100
## 2       -1.2438533  -0.7475604            -1.4506553   -1.0398394    -
0.2639581
## 3        0.4835876   0.4671768            -0.0991015    0.2726425    -
0.8962365
## 4        0.5105645   0.7104138            -0.4106751    0.1259246    -
1.2053545
## 5       -0.6256364  -0.7884179            -1.2726874   -0.2155274    -
0.7322711
## 6       -1.1746253  -0.3617901            -1.4447498   -1.2385764
0.9316561
##   degree_spondylolisthesis
## 1               -0.7069165
## 2               -0.5786208
## 3               -0.7941370
## 4               -0.4016390
## 5               -0.4893150
## 6               -0.6407526
```

Now, we need to convert our **class** variable to a factor and save it in its own variable.

```
str(column_df$class)
```

```
##  chr [1:310] "Abnormal" "Abnormal" "Abnormal" "Abnormal" "Abnormal" ...
```

```
column_df$class <- as.factor(column_df$class) #convert categorical variable
to factor
class <- column_df$class
```

Using our scaled data and our factor variable, we can create our final data frame that will be used in our SVM.

```
column_final <- cbind(column_scale, class) #combine response variable and
scaled variables
head(column_final)
```

```
##   pelvic_incidence pelvic_tilt lumbar_lordosis_angle sacral_slope
pelvic_radius
## 1        0.1468489   0.5005594            -0.6641032   -0.1846517    -
1.4453100
## 2       -1.2438533  -0.7475604            -1.4506553   -1.0398394    -
0.2639581
## 3        0.4835876   0.4671768            -0.0991015    0.2726425    -
0.8962365
## 4        0.5105645   0.7104138            -0.4106751    0.1259246    -
1.2053545
## 5       -0.6256364  -0.7884179            -1.2726874   -0.2155274    -
0.7322711
## 6       -1.1746253  -0.3617901            -1.4447498   -1.2385764
0.9316561
```

```
##   degree_spondylolisthesis    class
## 1               -0.7069165 Abnormal
## 2               -0.5786208 Abnormal
## 3               -0.7941370 Abnormal
## 4               -0.4016390 Abnormal
## 5               -0.4893150 Abnormal
## 6               -0.6407526 Abnormal
```

```
str(column_final)
```

```
## 'data.frame':    310 obs. of  7 variables:
##  $ pelvic_incidence        : num [1:310, 1] 0.147 -1.244 0.484 0.511 -
0.626 ...
##   ..- attr(*, "scaled:center")= num 60.5
##   ..- attr(*, "scaled:scale")= num 17.2
##  $ pelvic_tilt             : num [1:310, 1] 0.501 -0.748 0.467 0.71 -0.788
...
##   ..- attr(*, "scaled:center")= num 17.5
##   ..- attr(*, "scaled:scale")= num 10
##  $ lumbar_lordosis_angle   : num [1:310, 1] -0.6641 -1.4507 -0.0991 -
0.4107 -1.2727 ...
##   ..- attr(*, "scaled:center")= num 51.9
##   ..- attr(*, "scaled:scale")= num 18.6
##  $ sacral_slope            : num [1:310, 1] -0.185 -1.04 0.273 0.126 -
0.216 ...
##   ..- attr(*, "scaled:center")= num 43
##   ..- attr(*, "scaled:scale")= num 13.4
##  $ pelvic_radius           : num [1:310, 1] -1.445 -0.264 -0.896 -1.205 -
0.732 ...
##   ..- attr(*, "scaled:center")= num 118
##   ..- attr(*, "scaled:scale")= num 13.3
##  $ degree_spondylolisthesis: num [1:310, 1] -0.707 -0.579 -0.794 -0.402 -
0.489 ...
##   ..- attr(*, "scaled:center")= num 26.3
##   ..- attr(*, "scaled:scale")= num 37.6
##  $ class                   : Factor w/ 2 levels "Abnormal","Normal": 1 1 1
1 1 1 1 1 1 1 ...
```

To create our training and testing data sets, we will use the **createDataPartition()** function, where we will partition 70% of the data into the training data set and 30% into the training data set.

```
set.seed(789)
index <- createDataPartition(column_final$class, p =0.7, list = FALSE)
#Train set
columnTrain <- column_final[index,] #index for training set
train_labels <- column_final[7][index,] #labels for training
cat("Dimensions of Training Set:", dim(columnTrain))
```

```
## Dimensions of Training Set: 217 7
```

```
#Test set
columnTest <- column_final[-index,] #not index for test set
test_labels <- column_final[-index,7] #labels for test
cat("Dimensions of Testing Set:", dim(columnTest))

## Dimensions of Testing Set: 93 7
```

Now, we are ready to build our classifier. To do this, we will use the **ksvm()** function.

```
svm_linear_model <- ksvm(class ~ ., data = columnTrain, kernel =
"vanilladot") #SVM Linear Model

##  Setting default kernel parameters

svm_linear_model

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 83
##
## Objective Function Value : -71.993
## Training error : 0.147465
```

Using our model above, we will apply various performance measures.

Part 1

The first measure we will use is Cohen's kappa. Cohen's kappa is used to measure the agreement of two raters or methods rating on categorical scales (Kassambara, n.d.). "This process of measuring the extent to which two raters assign the same categories or score to the same subject is called *inter-rater reliability* (Kassambara, n.d.)." Cohen's kappa can be used for two categorical variables, which can be either two nominal (feature with categories) or two ordinal (ordered categories) variables. There are other variants of Cohen's Kappa. One includes the weighted kappa, which can only be used for ordinal variables (Kassambara, n.d.). Light's kappa is the average of all possible two-raters Cohen's kappa when there are more than two categorical variables. The Fleiss kappa is an adaptation of Cohen's kappa for n raters, where $n > 2$ (Kassambara, n.d.).

Kappa values range from 0 to a maximum of 1, which indicates a perfect agreement between the model's predictions and the true values (Lantz, 2015).

- Poor agreement = less than 0.20
- Fair agreement = 0.20 to 0.40
- Moderate agreement = 0.40 to 0.60

- Good agreement = 0.60 to 0.80
- Very good agreement = 0.80 to 1.00

The following formula is used to calculate Cohen's kappa:

$$\kappa = \frac{Pr(a) - Pr(e)}{1 - Pr(e)}$$

where $Pr(a)$ is the proportion of the actual/observed agreement and $Pr(e)$ is the proportion of chance agreement, or the expected agreement between the classifier and the true values (Lantz, 2015). These proportions can easily be obtained from a confusion matrix, or a two-way cross-tabulation. To create this table, we can use the **CrossTable()** function with our model's predictions and our test labels as the arguments.

```
linear_pred <- predict(svm_linear_model, columnTest)
CrossTable(test_labels, linear_pred, dnn = c("Actual","Predicted" ))
```

```
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  93
##
##
##              | Predicted
##       Actual |  Abnormal |    Normal | Row Total |
## -------------|-----------|-----------|-----------|
##     Abnormal |        50 |        13 |        63 |
##              |     6.911 |     8.392 |           |
##              |     0.794 |     0.206 |     0.677 |
##              |     0.980 |     0.310 |           |
##              |     0.538 |     0.140 |           |
## -------------|-----------|-----------|-----------|
##       Normal |         1 |        29 |        30 |
##              |    14.512 |    17.622 |           |
##              |     0.033 |     0.967 |     0.323 |
##              |     0.020 |     0.690 |           |
##              |     0.011 |     0.312 |           |
## -------------|-----------|-----------|-----------|
## Column Total |        51 |        42 |        93 |
##              |     0.548 |     0.452 |           |
## -------------|-----------|-----------|-----------|
##
```

Using the table above, we can calculate the proportions of the observed agreement ($Pr(a)$). To do this, we add the proportion of all instances where the predicted type and actual class type agree (Lantz, 2015). To calculate the expected agreement ($Pr(e)$), we need to calculate the probability that the chance alone would lead the predicted and actual values to match (Lantz, 2015). First, we need to find the probability that both actual and predicted values are *Abnormal*. This is calculated using the equations below.

$$P(actual\ type\ is\ Abnormal) * P(predicted\ type\ is\ Abnormal).$$

Then, we need to find the probability that both actual and predicted values are *Normal*. This is calculated using the equations below.

$$P(actual\ type\ is\ Normal) * P(predicted\ type\ is\ Normal).$$

These values can be obtained from the row or column totals in the two-way cross-tabulation above (Lantz, 2015).

1. $P(actual\ type\ is\ Abnormal) = 0.548$
2. $P(predicted\ type\ is\ Abnormal) = 0.677$
3. $P(actual\ type\ is\ Normal) = 0.452$
4. $P(predicted\ type\ is\ Normal) = 0.323$

Finally, to calculate $Pr(e)$ we add both products (Lantz, 2015). In other words,

$$Pr(e)$$
$$= [P(actual\ type\ is\ Abnormal) * P(predicted\ type\ is\ Abnormal)]$$
$$+ [P(actual\ type\ is\ Normal) * P(predicted\ type\ is\ Normal)].$$

Using these values, we can calculate the kappa statistic.

```
pr_a <- 0.538 + 0.312
pr_e <- (0.6770*.548)+(0.323*0.452)
k <- (pr_a-pr_e)/(1-pr_e) #calculate Kappa statistic
round(k, 4)

## [1] 0.6894
```

This value indicates that we have good agreement between the model's predictions and the true values.

To simplify this process, we can use the **Kappa()** function in the **vcd** package.

```
pred_table <- table(test_labels, linear_pred)
Kappa(pred_table)

##             value     ASE     z  Pr(>|z|)
## Unweighted 0.6882 0.07396 9.305 1.344e-20
## Weighted   0.6882 0.07396 9.305 1.344e-20
```

Notice, that our manual calculation agrees with our **Kappa()** output, other than a small difference due to rounding. Also, notice there are two kappa values: unweighted and

weighted. In cases of a two-outcome event, such as *Abnormal* and *Normal*, the weighted and unweighted kappa statistics will be identical (Lantz, 2015). The unweighted kappa corresponds to the Cohen's Kappa. The weighted kappa should be considered for ordinal variables or when there are varying degrees of agreement (Lantz, 2015).

Part 2

The next performance measures we are going to analyze are precision and recall. Precision, also known as the positive predictive value, is defined as the proportion of positive examples that are truly positive (Lantz, 2015). In other words, when a model predicts the positive class, how often is it correct (Lantz, 2015)? Precision is best used when the costs of False Positive are high (Shung, 2020). Consider a model that was imprecise; over time, the results would be less likely to be trusted. The following is the formula for precision (Shung, 2020):

$$Precision = \frac{True\ Positive}{True\ Positive\ +\ False\ Positive}.$$

This can be simplified to,

$$Precision = \frac{True\ Positive}{Total\ Predicted\ Positive}.$$

The confusion matrix and its parts can be found below.

		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

Confusion Matrix (Shung, 2020)

Thus, using the CrossTable created in *Part 1*, we can calculate the precision of our model.

```
#Precision (Positive Predictive Value)
ppv <- 29/(29+13)
round(ppv, 4)
```

```
## [1] 0.6905
```

Similarly, we can simplify this process by using the **posPredValue()** function in the **caret** package.

```
posPredValue(test_labels, linear_pred,  positive = "Abnormal")

## [1] 0.7936508
```

Alternatively, recall is the measure of how complete the results are (Lantz, 2015). It should be noted that recall and sensitivity (also know as the true positive rate) utilize the same formulas, but is the measure of the proportion of positive examples that were correctly classified. A model with a high recall captures a large portion of the positive examples, which makes recall best used when there is a high cost associated with False Negative (Shung, 2020). The formula for recall, and sensitivity, is (Lantz, 2015):

$$Recall = \frac{True\ Positive}{True\ Positive\ +\ False\ Negative}.$$

This can be simplified to,

$$Recall = \frac{True\ Positive}{Total\ Actual\ Positive}.$$

Thus, using the CrossTable created in *Part 1*, we can calculate the recall of our model.

```
#Recall (Also Sensitivity): TP/TP+FN
  #Sensitivity (True Positive Rate) = TP/TP+FN

tpr <- 29/(29+1)
tpr

## [1] 0.9666667
```

Similarly, we can simplify this process by using the **sensitivity()** function in the **caret** package.

```
sensitivity( test_labels,linear_pred, positive = "Abnormal")

## [1] 0.9803922
```

Ideally, a model will have high precision and high recall because that indicates that all results were labeled correctly (Precision-Recall, n.d.). A model with high precision but low recall indicates that most of the predicted labels are correct, but a model with high recall but low precision indicates that most of the predicted labels are incorrect (Precision-Recall, n.d.). From the results above, we have a model with moderate precision, but high recall. This indicates that of our predicted positive values, many are not actually positive (precision). and the recall shows that only many of the actual positives our model captured were labeled as positive.

Part 3

In the final part of our assignment, we will use the F-Measure (also known as the F1 Score) and specificity. First, the F-Measure combines precision and recall using the *harmonic mean*. The harmonic mean is a type of average that is used for rates of change and is used because both precision and recall are expressed as proportions, which can be interpreted as rates (Lantz, 2015). The F-Measure is a better measure if we need to seek a balance between precision and recall and there is an uneven class distribution (large number of Actual Negatives) (Shung, 2020). A high F-Measure value indicates high precision and recall. The formula for the F-Measure is (Shung, 2020):

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}.$$

Thus, we can calculate the F-Measure for our model.

```
#F-Measure =
f_measure = 2*((ppv*tpr)/(ppv+tpr))
f_measure

## [1] 0.8055556
```

Specificity, also known as the true negative rate, measures the proportion of negative examples that were correctly classified (Lantz, 2015). The formula for specificity is,

$$Specificity = \frac{True\ Negative}{True\ Negative\ +\ False\ Positive}.$$

```
#Specificity (True Negative Rate) = TN/TN+FP
tnr <- 50/(50+13)
tnr

## [1] 0.7936508
```

We can use the **specificity()** function in the **caret** package.

```
specificity(test_labels, linear_pred,  positive = "Abnormal")

## [1] 0.6904762
```

To simplify the equations above, we can use the **confusionMatrix()** function in the **caret** package and use the parameter **mode = "everything"** to obtain the cross table and the performance measures above.

```
confusionMatrix(test_labels, linear_pred,  mode = 'everything', positive =
"Abnormal")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
```

```
##   Abnormal       50     13
##   Normal          1     29
##
##                Accuracy : 0.8495
##                  95% CI : (0.7603, 0.9152)
##     No Information Rate : 0.5484
##     P-Value [Acc > NIR] : 6.716e-10
##
##                   Kappa : 0.6882
##
##  Mcnemar's Test P-Value : 0.003283
##
##             Sensitivity : 0.9804
##             Specificity : 0.6905
##          Pos Pred Value : 0.7937
##          Neg Pred Value : 0.9667
##               Precision : 0.7937
##                  Recall : 0.9804
##                      F1 : 0.8772
##              Prevalence : 0.5484
##          Detection Rate : 0.5376
##    Detection Prevalence : 0.6774
##       Balanced Accuracy : 0.8354
##
##        'Positive' Class : Abnormal
##
```

Notice, additional performances are also included in the performance matrix.

Conclusion

For this assignment, created a Support Vector Machine classifier using orthopedic patient data. Using this model, we explored various performance measures such as Cohen's kappa, precision, recall, F-Measure, and Specificity. We use other performance measures because accuracy doesn't give a comprehensive view of the model. The accuracy rate for our linear SVM was ~85% and an error rate of ~15%. After calculating our kappa statistic, we found that we have a "good agreement" between the model's predictions and true values. Further, our model had a moderate precision value and a high recall value. This indicates that our model predicted most of the labels incorrectly. We also found a low specificity value which indicates that many negative values were incorrectly classified. Finally, our F-Measure value was between a moderate and high level, indicating a high precision and recall for our SVM model.

Resources

Kassambara, A. (n.d.). Cohen's kappa in r. Datanovia. Retrieved November 1, 2020, from https://www.datanovia.com/en/lessons/cohens-kappa-in-r-for-two-categorical-variables/

Lantz, B (2015). Machine Learning with R - second Edition, Packt Publishing. (Chapter 10)

Precision-Recall . (n.d.). Retrieved November 1, 2020, from https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#:~:text=A%20system%20with%20high%20precision,with%20all%20results%20labeled%20correctly.

Scale function. (n.d.). Retrieved November 1, 2020, from https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/scale

Scaling numeric values only in a dataframe with mixed types. (n.d.). Stack Overflow. Retrieved November 1, 2020, from https://stackoverflow.com/questions/23619188/r-scaling-numeric-values-only-in-a-dataframe-with-mixed-types

Shung, K. P. (2020, April 10). Accuracy, precision, recall or f1? Medium. https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9

Vertebral column dataset. (n.d.). Retrieved November 1, 2020, from https://kaggle.com/caesarlupum/vertebralcolumndataset

World Class From the Expert Week 2