Machine Learning Classification Algorithms

Jessica Young
University of Notre Dame
Center for Social Science Research
jyoung22@nd.edu
Overview

- Trees
- Random Forest
- Discriminant Analysis Overview
- LDA/QDA
- SVM
Traditionally, logistic regression can be used for classification problems, while the model produced is easy to interpret it is also somewhat inflexible. Machine learning methods can instead be used for when a more flexible model is needed at the cost of interpretability.

While models like trees still allow for some interpretability, support vector machines (SVM) and neural nets (NN) allow for next to none due to their “black box” nature.
Decision trees partition the input space into $q$ disjoint regions such that:

$$\mathcal{X} = R_1 \cup R_2 \cup \cdots \cup R_q = \bigcup_{l=1}^{q} R_l$$

The prediction in each region in $R_l$ consists of a single constant, $f(x)$, which makes trees a piece-wise constant function estimator.
# simulate a data set with one predictor variable

```r
n = 200  # sample size
x = seq(-10, 10, length=n)  # x values
f = function(x)
  {
    # create the sine function for y values
    z = sin(x)/x
    z[is.nan(z)] = 1  # replace NA values with 1
    return(z)
  }

# add random error to y values
y = f(x) + rnorm(n = n, mean = 0, sd = .1)
xy = data.frame(x, y)  # store as data frame
```
Now let’s run CART on our toy data set and see how well it can predict.

```r
# CART for regression
rpart_output = rpart(y~x, data=xy, method='anova')
# prediction
yhat = predict(rpart_output, data.frame(x = x))
# MSE
mean((y-yhat)^2)
```

```r
## [1] 0.01684804
```
While see the MSE is useful, CART also lets you visual the decisions it makes when predicting. In the next slide, we’ll see the major decisions CART made for each observation when determining the predicted value.
Regression Tree on Simulated Data

- $x \geq 2.814$
- $x < 6.432$
- $x < -2.312$
- $x \geq -6.231$
- $x < -3.116$
- $x \geq 1.91$
- $x < -1.407$

- $-0.1144 \quad 0.07616$
- $-0.1664 \quad 0.1738$
- $0.09544 \quad 0.377$
- $0.5233 \quad 0.8715$
Other ways to visualize the data include scatter-plots. In the next slide, we can see the raw data (black circles) with a smoothed line (red dashes) to show the patterns in the data, and how our CART algorithm did (green dashes). This plot is useful for seeing how close the CART algorithm actually came to learning all of the patterns in the data.

Keep in mind though- if your final model follows the smoothed line near perfect, then you’re most likely overfitting and will do poorly when new data is added to the mix!
Example
In random forests, numerous trees are created and then aggregates across all the trees created to determine the classification of each observation.

\[
\hat{f}_{RF}^{(B)}(x_{\text{new}}) \arg\max_{y \in Y} \left\{ \frac{1}{B} \left\{ I(\hat{f}^{(b)}(x_{\text{new}}) = y) \right\} \right\}
\]
Random Forest

1. Draw a bootstrap sample from the data
2. Build the $b^{th}$ tree, $\hat{f}^{(b)}(\cdot)$
   - at each node, only $q$ variables are considered (where there are a total of $p$ variables and $q \ll p$ and $q$ was selected a priori)
   - Select the best split $(x_{jk}, \tau_k), k = 1, \ldots, q$ out of the $q$ selected variables
   - Do not prune the tree
   - Store $\hat{f}^{(b)}(\cdot)$

Aggregate predictions of the ensemble
Balanced Random Forest

for ($b = 1$ to $B$)

1. Draw a bootstrap sample from the minority class with replacement and then randomly draw the same amount from the majority class
2. Build all the $b$ tree, $\hat{f}^{(b)}(\cdot)$
3. Aggregate predictions (Chen, Liaw, and Breiman 2001)
Goal is to place weights on the class priors of the response in the training set such that these class weights penalize misclassifying the minority class more than the majority class.

At each terminal node, predictions are made by weighted majority vote.

Final class prediction is made by aggregating these weighted votes in each individual tree (Chen, Liaw, and Breiman 2001)
LDA

- attempts to classify data by separating it with a linear plane
- assumes data is created from two different normal distributions with equal co-variances
- models each cluster separately to have different means
- figures out the optimal place to put a line between the two distributions to minimize error
- is robust to departure from normality
If the departure from normality is severe, then QDA (quadratic discriminant analysis) can be used instead. QDA uses a quadratic function to create the separating plane between the two classes. Anywhere LDA works, QDA will also work. However, due to creating a quadratic plane, QDA will take much longer to run.
Support vector machines aims to classify data by creating a linear boundary between the different classes of the response. It does this by first placing a linear separator and optimizing the placement such that the margin around it is as big as possible.
SVM
For nonlinear data problems, kernel SVM can be used instead. The non-linearity issue is avoided by using a kernel (similarity measure).

Common kernels are:

**RBF**

\[
K(x_i, x_j) = \exp(-\delta \|x_i - x_j\|^2_2)
\]

\(\delta \equiv \text{bandwidth}\)

**Laplace**

\[
K(x_i, x_j) = \exp(-\delta \|x_i - x_j\|_1)
\]

\(\delta \equiv (\text{usually}) \ p^{-1}\)
Let’s see how these algorithms we just discussed works with another data set. Before we get into making any models though, we’ll first plot the data to see what out looks like.

From the plot, we can see that the data looks like an x. Depending on how you view the data, you could say there’s either 2 or 4 distinct classes here. Can you figure out why?
# run lda on the news data set

```r
lda_news = lda(y~., data = news)
lda_yhat = predict(lda_news, news[, -3])$class

confusionMatrix(news[, 3], lda_yhat)$table
```

<table>
<thead>
<tr>
<th></th>
<th>Reference</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>152</td>
<td>148</td>
</tr>
<tr>
<td>1</td>
<td>149</td>
<td>151</td>
</tr>
</tbody>
</table>

Given that the scatter-plot appeared to suggest non-linearity, it makes sense that LDA does a poor job at classification for this data.
### Example

```r
# run qda on the news data set
qda_news = qda(y~., data = news)
qda_yhat = predict(qda_news, news[,,-3])$class

confusionMatrix(news[,3], qda_yhat)$table
```

<table>
<thead>
<tr>
<th>Reference</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>294</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>295</td>
</tr>
</tbody>
</table>

And since the data is non-linear, QDA appears to do very well with classification.
But running though the data once as a whole doesn’t give an accurate representation of how the algorithms will actually fair. After all, training an algorithm with an entire data set sets you up to overfit! In the next few slides, we’ll look at code you could run to better assess these algorithms fit of the data.
We’ll start off by splitting the data into training and test sets.

\[
R = 100 \ # \ 100 \ reps
\]

\[
\text{err\_mat\_news} = \text{matrix}(0, \text{nrow}=R, \text{ncol}=3)
\]

\[
\text{for}(r \ in \ 1:R)\
\]

\[
\text{id} = \text{holdout}(\text{news}\$y, \text{ratio}=.7)
\]
\[
\text{tr\_xy} = \text{news}[\text{id}\$tr,]
\]
\[
\text{te\_xy} = \text{news}[\text{id}\$ts,]
\]
\[
\text{te\_y} = \text{te\_xy}[,3]
\]

\[
\text{class0} = \text{subset}(\text{tr\_xy}, \text{y==0})[,,-3]
\]
\[
\text{class1} = \text{subset}(\text{tr\_xy}, \text{y==1})[,,-3]
\]
Here we’ll run LDA and QDA.

# lda
   lda_news = lda(y~., data = tr_xy)
   lda_yhat = predict(lda_news, te_xy[, -3])$class

# qda
   qda_news = qda(y~., data = tr_xy)
   qda_yhat = predict(qda.news, te_xy[, -3])$class
For C-SVM, we’ll optimize hyperparameters using CV.

```r
# c-svm
# finding opt c
# initialize value sequence and matrix

c_vec = seq(2^(-7), 2^7, length.out = 500)
c_opt = matrix(0, ncol=2, nrow=length(c_vec))
c_opt[,1] = c_vec # store values in first column

for(i in 1:length(c_vec)){
  c_opt[i,2] = cross(ksvm(y~., data = tr_xy, 
                       type = 'C-svc', # C-SVM
                       cross = 10, # cv 10 times
                       C = c_vec[i])) }

# optimal value

# optimal value

c_opt_fin =
  c_opt[which(c_opt[,2]==min(c_opt[,2])), 1]
```
Using the optimized hyperparameters, we can run C-SVM now.

```r
# svm - final model with optimized hyperparameters
svm_news = ksvm(y~., data = tr_xy, type = 'C-svc',
               cross = 10, C = c_opt_fin)
svm_yhat = predict(svm_news, te_xy[,-3])

err_mat_news[r,1] = mean(te_y != lda_yhat)
err_mat_news[r,2] = mean(te_y != qda_yhat)
err_mat_news[r,3] = mean(te_y != svm_yhat)

cat('Finished rep',r,'\n')
}
```
Below is a chunk of code that finds the average error, the median error, and the standard deviation of error for each of the algorithms.

```r
colnames(err_mat_news) = c('LDA', 'QDA', 'SVM')

# store mean, median, and sd of errors
T1 = as.data.frame(cbind(c('Average', 'Median', 'Std Dev'),
                         apply(rbind(apply(err_mat_news, 2, mean),
                                  apply(err_mat_news, 2, median)),
                                 apply(err_mat_news, 2, sd)),
                         function(x) round(x,4)))
```
```r
print(T1)

## V1   LDA  QDA   SVM
## 1 Average 0.5245 0.0198 0.0212
## 2 Median   0.5278 0.0222 0.0222
## 3 Std Dev  0.027  0.009  0.0095
```

From the table we calculated, we can see that LDA does indeed fit the data poorly. On the other hand, QDA and SVM both do well and seem to fit the data nearly the same.
Graphs can also be used to compare and contrast the models created. Below is how ggplot2 can be used to make comparative box-plots of the data.

```r
# melt for ggplot
news_fin = melt(err_mat_news)
colnames(news_fin) = c('Rep_No','Method','Error')

# boxplot to show comparative results
ggplot(news_fin,
       mapping = aes(x = Method,
                      y = Error,
                      fill = Method))+
  ggttitle("News") +
  geom_boxplot()
```
Example
From the box-plots in the last slide, we can see that even though QDA and SVM appeared equal in the table we calculated, there is in fact a difference in how the algorithms worked. The box-plots show that although the average error is around the same, QDA’s average error is actually skewed on to the lower side. This means that it’s pretty likely the error QDA produces from this data set could be lower than the average error.