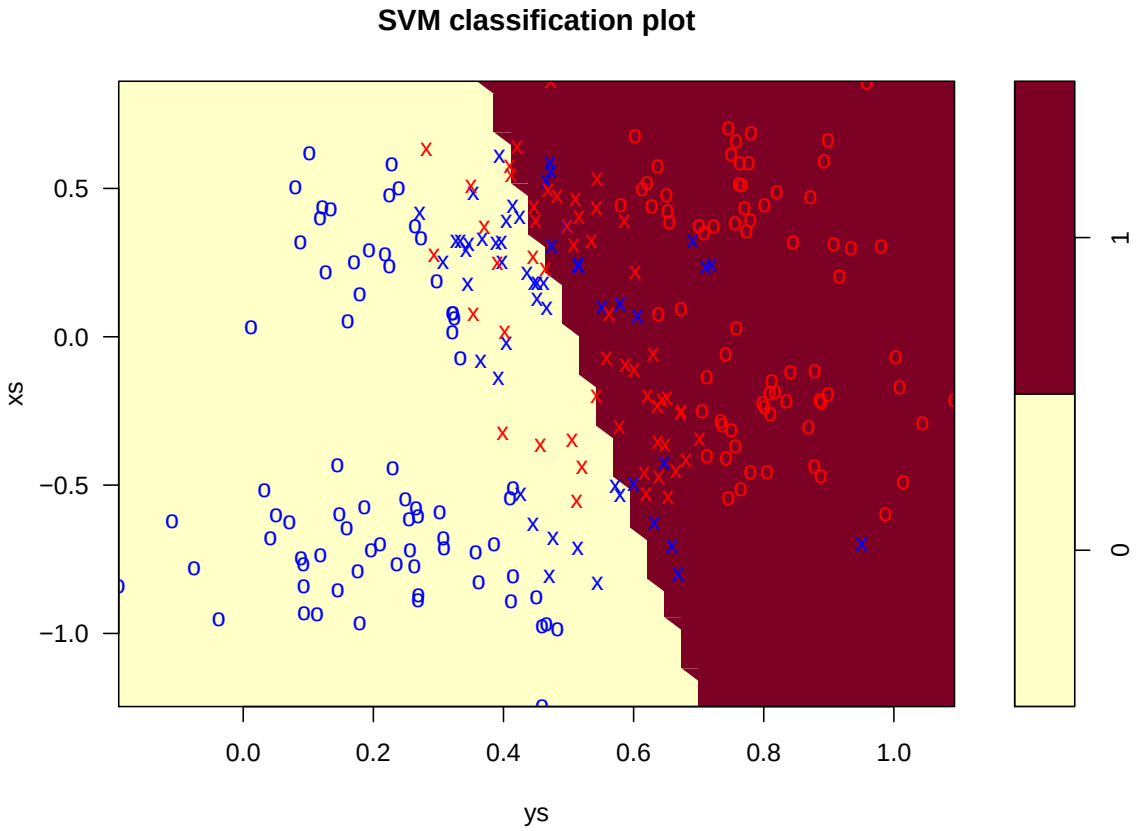
Homework 3

Ngoc Duong

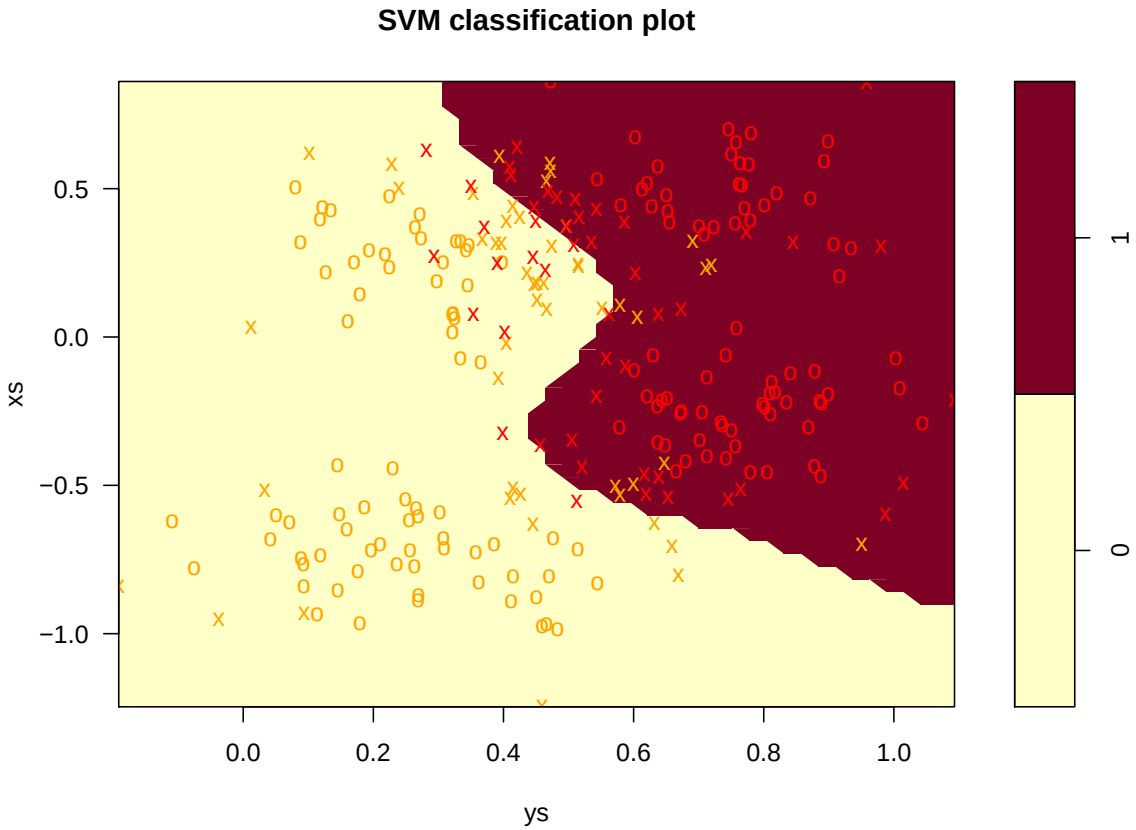
11/7/2020

Question 2

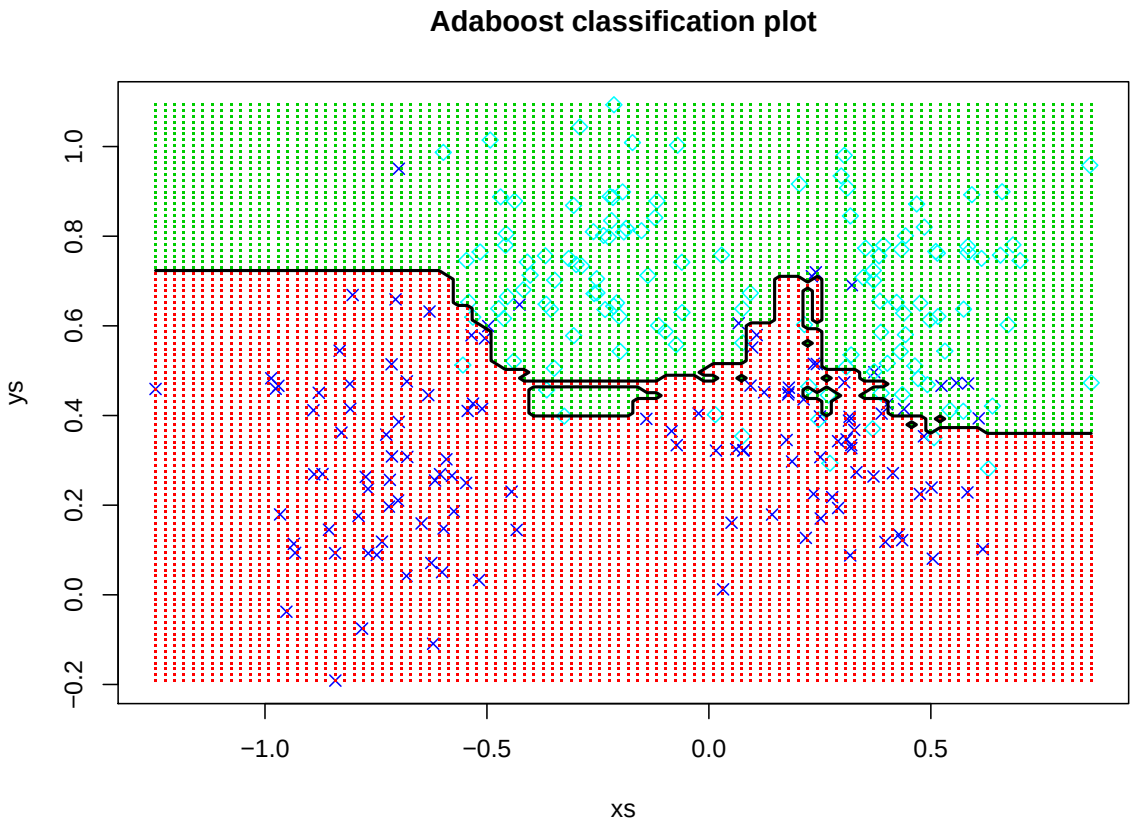
SVM linear classifier plot



SVM radial kernel plot



	Linear.classifier	Radial.kernel	Adaboost
Test error rate	0.1030000	0.0900000	0.1050000
Standard error	0.0096168	0.0090544	0.0096989



Discussion:

The three models have similar test prediction performance. The SVM model with radial kernel has the best performance on the test set (error rate 9%), compared with Adaboost (10.5%) and SVM with a linear classifier (10.3%). The standard error of the misclassification rate is also lowest for the SVM model with the radial kernel. We can see from the plot that the classes are mixed on a 2D plane with a linear boundary. Using radial kernel maps the data onto higher-dimensional space where the classes can be separated more easily, which might explain the lower error rate. Both SVM models were tuned using a grid of cost parameters (linear classifier) and cost and sigma (radial kernel) so the reported performance was from the best tuned models. Adaboost model was not tuned but set to have stump as weak learner and 50 boosting iterations, so performance might improve with more tuning.

In terms of the visualization of the models, we can see that the linear boundary for the SVM linear classifier is linear in the 2D plane, while the SVM radial kernel's separating plane from a higher-dimensional space translates to a non-linear boundary in 2D plane. AdaBoost used decision stumps to learn the classifier so the non-linear/non-smooth decision boundary shows this slow-learning pattern (reweighting the training points and adding one weak learner at a time).

Question 3

Here, I fit a neural network with 1 hidden layer and 10 nodes in this layer.

	Neural.net	Classification.tree
Test error rate	0.0768	0.0931

Based on the classification tree in **section 9.2.5**, we can calculate the error rate of the tree as the proportion of misclassifications out of 1536 emails being classified. This is calculated as:

$(19+1+37+1+9+3+14+3+9+16+6+18+7)/1536 = 143/1536 = 0.093$.

The test error rate of the neural network with one hidden layer and 10 units in this layer is 0.0768. In this case, we can see that neural network has very good performance and performs better than the tree on test data, despite not by a large margin. However, trees usually have small bias and large variance, so trees can classify training data well but have more inaccuracies on unseen data.

In terms of interpretability, trees are easy to interpret if they are not too big, and their high interpretability can be useful in many settings like clinical decision making. The neural network is a black-box model, which makes it difficult to explain the outcome/process compared to trees. Additionally, small changes in the specification of the neural network (number of hidden layers/within-layer nodes or activation function) may lead to different outcomes; therefore, we should take into account these characteristics given the the purpose and setting of the prediction task.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(tidyverse)
library(janitor)
library(e1071)
library(ModelMetrics)
library(caret)
library(microbenchmark)
library(glmnet)
library(broom)
library(mgcv)
library(data.table)
library(RColorBrewer)
library(gbm)
library(nnet)
library("grid")
library("gridExtra")
library("ada")

knitr::opts_chunk$set(
    echo = TRUE,
    warning = FALSE,
    fig.width = 8,
  fig.height = 6,
  out.width = "90%"
)
theme_set(theme_minimal() + theme(legend.position = "bottom"))
## Decision plot function
decisionplot <- function(model, data, class,
  resolution = 100, showgrid = TRUE, ...) {
  cl <- data$class
  data <- data[,1:2]
  k <- length(unique(cl))
  # make grid
  r <- sapply(data, range, na.rm = TRUE)
  xs <- seq(r[1,1], r[2,1], length.out = resolution)
  ys <- seq(r[1,2], r[2,2], length.out = resolution)
  g <- cbind(rep(xs, each=resolution), rep(ys, time = resolution))
```

```
  colnames(g) <- colnames(r)
  g <- as.data.frame(g)
  ### use predict function (class here)
  p <- predict(model, g)
  if(showgrid) points(g, col = as.integer(p)+1L, pch = ".")
  z <- matrix(as.integer(p), nrow = resolution, byrow = TRUE)
  contour(xs, ys, z, add = TRUE, drawlabels = FALSE,
    lwd = 2, levels = (1:(k-1))+.5)
  invisible(z)
}
#read in ripley data
train = fread("./PRNN/SYNTH.TR") %>% mutate(yc = factor(yc))
test = fread("./PRNN/SYNTH.TE") %>% mutate(yc = factor(yc))
#a) Construct a linear support vector classifier - use e1071
set.seed(2020)
svml.fit <-tune.svm(yc ~., data = train, kernel = "linear", preProc = c("scale", "center"),
                    cost =exp(seq(-5,-1,len=20)))
best.linear = svml.fit$best.model #get tuned model
pred.linear = predict(best.linear, test[,-3]) #prediction on test set

#error on the test set and standard error of this test error statistic
err.linear = mean(pred.linear != test$yc)
se.err.linear = sd(abs(as.numeric(pred.linear) - as.numeric(test$yc)))/sqrt(nrow(test))

plot(best.linear, train, symbolPalette = c("blue", "red"))
#b) Construct a support vector classifier with Radial Kernel
set.seed(2020)
svmr.fit <-tune.svm(yc ~., data = train,
                    kernel = "radial", preProc = c("scale", "center"),
                    cost =exp(seq(-5,0,len=10)),
                    gamma = exp(seq(-7,0, len=10)))
best.radial = svmr.fit$best.model #get best model
pred.radial = predict(best.radial, test[,-3]) #prediction on test set

#error on the test set and standard error of this test error statistic
err.radial = sum((pred.radial != test$yc))/length(test$yc)
se.err.radial = sd(abs(as.numeric(pred.radial) - as.numeric(test$yc)))/sqrt(nrow(test))

plot(best.radial, train, symbolPalette = c("orange", "red"))
#c) Construct a classifier using Adaboost with 50 boosting iterations
set.seed(2020)
bst = ada(yc~., train,
          iter=50, rpart.control(maxdepth=1))

pred.bst = predict(bst, test)
#error on the test set and standard error of this test error statistic
err.bst = mean(pred.bst != test$yc)
se.err.bst = sd(abs(as.numeric(pred.bst) - as.numeric(test$yc)))/sqrt(nrow(test))

err_tibble = data.frame(`Linear classifier` = c(err.linear, se.err.linear),
                    `Radial kernel` = c(err.radial, se.err.radial),
                    `Adaboost` = c(err.bst, se.err.bst))
rownames(err_tibble) = c("Test error rate", "Standard error")
```

```r
#result
err_tibble %>% knitr::kable()

# decision boundry
plot(train[,1:2], col = as.numeric(train$yc)+3L, pch = as.numeric(train$yc)+3L, main = "Adaboost classi
par(new=TRUE)
decisionplot(bst, train, class = yc)
#spam data
spam = read.csv("SPAM.csv")
spam_train = spam %>% filter(testid == "FALSE") %>% select(-testid) %>% mutate(spam = ifelse(spam == "T
spam_test = spam %>% filter(testid == "TRUE") %>% select(-testid) %>% mutate(spam = ifelse(spam == "TRU

set.seed(1)
unit = 10
nn = nnet(spam ~ ., data = spam_train, preProc = c("scale", "center"),
                 size=unit, skip=TRUE, MaxNWts=10000, trace=FALSE, maxit=100)
pred.nnet = predict(nn, spam_test[,-1], response = "prob")
pred.nnet = ifelse(pred.nnet > 0.5, 1, 0)

#error rate
nnet.err = mean(abs(as.numeric(pred.nnet) - (as.numeric(spam_test$spam)-1)))
#nnet.se = sd(as.numeric(pred.nnet) - as.numeric(spam_test$spam))/sqrt(nrow(spam_test))

#table for
err_tibble2 = data.frame(`Neural net` = round(nnet.err, 4),
                    `Classification tree` = round(143/1536, 4))
rownames(err_tibble2) = "Test error rate"

#result
err_tibble2 %>% knitr::kable()
```

(1) Suppose $X \in \mathbb{R}^p$ and $Y \in \{-1, 1\}$. Let $L(y, f(x))$ be the loss function. Let $f^* = \underset{f}{\operatorname{argmin}}\, E[L(y, f(x))]$

a) (Logistic regression) given $L(y, f(x)) = \log[1 + \exp(-yf(x))]$

Let $\pi(x) = P(Y=1 \mid X=x)$ and $\ell(yf(x)) = \log(1+\exp(-yf(x))) = L(y, f(x))$. We can rewrite the expected risk $E[L(Y, f(X))]$ in terms of $\pi(x)$ and $\ell(yf(x))$, and using the fact $y \in \{-1, 1\}$ as follows:

$$E[L(y, f(x))] = \int_y \int_x L(y, f(x)) P(x, y)\, dx\, dy = \int_y \int_x L(y, f(x)) P(y \mid x) P_x\, dx\, dy$$

$$= \int_x \ell(yf(x)) P(y \mid x) \Big|_{-1}^{1} P(x)\, dx$$

$$= \int_x \left[\ell(f(x)) P(y=1 \mid x) - \ell(-f(x)) P(y=-1 \mid x) \right] P(x)\, dx$$

$$= \int_x \left[\ell(f(x))\, \pi(x) - \ell(-f(x))\,(1-\pi(x)) \right] P(x)\, dx$$

We can take derivative of the function $[\ell(f(x))\pi(x) - \ell(-f(x))(1-\pi(x))]$ wrt $f(x)$ and set it equal to 0:

$$\frac{\partial \ell(f(x))\,\pi(x)}{\partial f(x)} - \frac{\partial \ell(-f(x))(1-\pi(x))}{\partial f(x)} = 0$$

$$\Leftrightarrow \frac{-\exp(-f(x))\,\pi(x)}{1+\exp(-f(x))} + \frac{\exp(f(x))(1-\pi(x))}{1+\exp(f(x))} = 0$$

$$\Leftrightarrow \frac{\exp(f(x))}{1+\exp(f(x))} + \pi(x)\, \frac{-\exp(-f(x)) - \exp(f(x)) - 2}{\exp(-f(x)) + \exp(f(x)) + 2} = 0$$

$$\Leftrightarrow \frac{\exp(f(x))}{1+\exp(f(x))} = \pi(x)$$

$$\Leftrightarrow f^*(x) = \underset{f}{\operatorname{argmin}}\, L(y, f(x)) = \log \frac{\pi(x)}{1-\pi(x)}$$

$$= \log \frac{P(y=1 \mid X=x)}{1-P(y=1 \mid X=x)} = \log \frac{P(y=1 \mid X=x)}{P(y=-1 \mid X=x)}$$

b) (SVM) given $L(y, f(x)) = [1 - yf(x)]_+$

Similarly to part a, we can take the expectation of the loss function over the joint distribution of X and Y, and use $y \in \{-1, 1\}$ to obtain:

$$E[(1-yf(x))_+ \mid X=x] = (1-f(x))_+ \pi(x) + (1+f(x))_+ (1-\pi(x))$$

- Consider $f(x) < -1 \Rightarrow E\left[(1-yf(x))_+ \mid X=x\right] = \left[1-f(x)\right]\pi(x)$
  we can see that with $\pi(x) \in [0,1]$, this function is monotone decreasing and $\min E = \lim_{f \to -1} E$
- Consider $f(x) > 1 \Rightarrow E\left[(1-yf(x))_+ \mid X=x\right] = \left[1+f(x)\right]\left[1-\pi(x)\right]$
  we can see that with $\pi(x) \in [0,1]$, this function is monotone increasing and $\min E = \lim_{f \to 1} E$

$\Rightarrow$ In both cases, the expected loss is not minimized over the above ranges of $f(x)$.
Plus, we can see that truncating at -1 and 1 gives a lower loss

- Consider $f(x) \in [-1, 1]$, which implies:

$$E\left[(1-yf(x))_+ \mid X=x\right] = (1-f(x))\pi(x) + (1+f(x))(1-\pi(x))$$
$$= (1-2\pi(x))f(x) + 1$$

To minimize this loss, consider two cases:

- $\pi(x) = P(y=1 \mid x) \geq \frac{1}{2} \Leftrightarrow 1-2\pi(x) \leq 0 \Leftrightarrow \pi(x) - \frac{1}{2} \geq 0$, which implies $f(x)$ should be >0 to minimize expected loss (specifically $f=1$)
- $\pi(x) = P(y=1 \mid x) < \frac{1}{2} \Leftrightarrow 1-2\pi(x) > 0 \Leftrightarrow \pi(x) - \frac{1}{2} < 0$, which implies $f(x)$ should be <0 to minimize expected loss (specifically $f=-1$)

Therefore, $f^*(x) = \text{sign}\left[P(y=1 \mid x) - \frac{1}{2}\right]$.

c) (Regression) given $L(y, f(x)) = \left[y - f(x)\right]^2$
we want to find the minimizer of the quadratic loss:

$$f^*(x) = \arg\min_f E\left[(y-f(x))^2\right]$$

From table 12.1 (text), we have $\left[y-f(x)\right]^2 = \left[1-yf(x)\right]^2$
Similar to part a, we can take derivative of the expected loss function wrt $f(x)$, and use the fact $y \in \{-1, 1\}$ to obtain:

$$-2\left[1-f^*(x)\right]\pi(x) + 2\left[1+f^*(x)\right]\left[1-\pi(x)\right] = 0$$
$$\Leftrightarrow -2\pi(x) + 2\pi(x)f^*(x) + 2f^*(x) - 2\pi(x)f^*(x) + 2 - 2\pi(x) = 0$$
$$\Leftrightarrow 2f^*(x) = 4\pi(x) - 2$$
$$\Leftrightarrow f^*(x) = 2\pi(x) - 1$$
$$= 2P(y=1 \mid x) - 1$$

Another way to get to the minimizer using Gauss Markov Theorem:
From the Gauss Markov theorem, we have that the minimizer of the squared error in this case quadratic loss) is the conditional expectation of y, $E[y \mid x]$
By definition of expectation, and using the fact $y \in \{-1, 1\}$, we have:

$$E[y \mid x] = \sum_{y \in \{-1,1\}} y\, P(y \mid x) = -1 \cdot P(y=-1 \mid x) + 1 \cdot P(y=1 \mid x)$$
$$= -1\left[1 - P(y=1 \mid x)\right] + P(y=1 \mid x)$$
$$= 2P(y=1 \mid x) - 1$$

d) we are given the loss function as $L(y, f(x)) = \exp[-yf(x)]$

we want to find the minimizer $f^*(x) = \arg\min_f E[\exp(-yf(x))]$

we can take the derivative of the expected loss function wrt $f(x)$ and set it to equal 0:

$$\frac{\partial E[\exp(-yf(x))]}{\partial f} = E[-y\exp(-yf(x))] = 0$$

Similar to a, when we rewrite the expected risk taken over the joint distribution of X and Y and using the information that $y \in \{-1, 1\}$, from the above equation, we have:

$$-(-1)\exp(-(-1)f(x))P(y=-1|x) - \exp(-f(x))P(y=1|x) = 0$$

$$\Leftrightarrow \exp(f(x))P(y=-1|x) - \exp(-f(x))P(y=1|x) = 0$$

Multiply both sides by $\exp(f(x))$ gives:

$$\exp(2f(x))P(y=-1|x) - P(y=1|x) = 0$$

$$\Leftrightarrow \exp(2f(x)) = \frac{P(y=1|x)}{P(y=-1|x)}$$

$$\Leftrightarrow f^*(x) = \frac{1}{2}\log\frac{P(y=1|X=x)}{P(y=1|X=x)}.$$