Lab 3

Amir ElTabakh

11:59PM March 4, 2021

Support Vector Machine vs. Perceptron

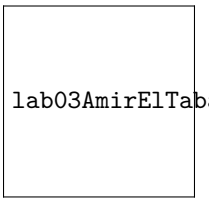
We recreate the data from the previous lab and visualize it:

```
pacman::p_load(ggplot2)
Xy_simple = data.frame(
 response = factor(c(0, 0, 0, 1, 1, 1)), #nominal
 first_feature = c(1, 1, 2, 3, 3, 4),    #continuous
 second_feature = c(1, 2, 1, 3, 4, 3)    #continuous
)

Xy_simple_response = as.numeric(Xy_simple$response)
Xy_simple_input = Xy_simple
Xy_simple_input$response = NULL



simple_viz_obj = ggplot(Xy_simple, aes(x = first_feature, y = second_feature, color = response)) +
  geom_point(size = 5)
simple_viz_obj
```



lab03AmirElTabakh_files/figure-latex/unnamed-chunk-1-1.pdf

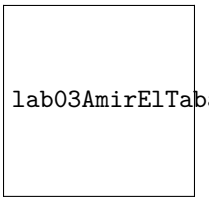
Use the `e1071` package to fit an SVM model to the simple data. Use a formula to create the model, pass in the data frame, set kernel to be `linear` for the linear SVM and don't scale the covariates. Call the model object `svm_model`. Otherwise the remaining code won't work.

```
pacman::p_load(e1071)
svm_model = svm(
  formula = Xy_simple$response~.,
  data = Xy_simple,
  kernel = "linear",
  scale = FALSE
)
```

and then use the following code to visualize the line in purple:

```
w_vec_simple_svm = c(
  svm_model$rho, #the b term
  -t(svm_model$coefs) %*% cbind(Xy_simple$first_feature, Xy_simple$second_feature)[svm_model$index, ] #
```

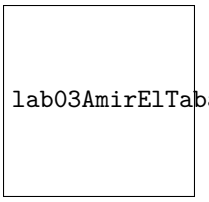
```
)
simple_svm_line = geom_abline(
    intercept = -w_vec_simple_svm[1] / w_vec_simple_svm[3],
    slope = -w_vec_simple_svm[2] / w_vec_simple_svm[3],
    color = "purple")
simple_viz_obj + simple_svm_line
```



Source the `perceptron_learning_algorithm` function from lab 2. Then run the following to fit the perceptron and plot its line in orange with the SVM's line:

```
#' TO-DO: Provide a name for this function
#'
#' TO-DO: Explain what this function does in a few sentences
#'
#' @param Xinput      A matrix of features for training data observations
#' @param y_binary    A vector of training data labels
#' @param MAX_ITER    Number of epochs to run to train the data
#' @param w           A vector with p+1 vectors
#'
#' @return            The computed final parameter (weight) as a vector of length p + 1
perceptron_learning_algorithm = function(Xinput, y_binary, MAX_ITER = 100000, w = NULL){
  Xinput = as.matrix(cbind(1, Xinput))
  p = ncol(Xinput)
  w = rep(0, p)
    for (iter in 1 : MAX_ITER){
      for (i in 1 : nrow(Xinput)){
        x_i = Xinput[i, ]
        yhat_i = ifelse(sum(x_i * w) > 0, 1, 0)
        y_i = y_binary[i]
        for (j in 1 : p) {
          w[j] = w[j]+(y_i - yhat_i)*x_i[j]
        }
      }
    }
  w
}
```

```
w_vec_simple_per = perceptron_learning_algorithm(
  cbind(Xy_simple$first_feature, Xy_simple$second_feature),
  as.numeric(Xy_simple$response == 1)
)
simple_perceptron_line = geom_abline(
    intercept = -w_vec_simple_per[1] / w_vec_simple_per[3],
    slope = -w_vec_simple_per[2] / w_vec_simple_per[3],
    color = "orange")
simple_viz_obj + simple_perceptron_line + simple_svm_line
```



lab03AmirElTabakh_files/figure-latex/unnamed-chunk-5-1.pdf

Is this SVM line a better fit than the perceptron?

The SVM line fits much better than the perceptron model.

Now write pseuocode for your own implementation of the linear support vector machine algorithm using the Vapnik objective function we discussed.

Note there are differences between this spec and the perceptron learning algorithm spec in question #1. You should figure out a way to respect the `MAX_ITER` argument value.

```
#' Support Vector Machine
#
#' This function implements the hinge-loss + maximum margin linear support vector machine algorithm of
#'
#' @param Xinput      The training data features as an n x p matrix.
#' @param y_binary    The training data responses as a vector of length n consisting of only 0's and 1'
#' @param MAX_ITER    The maximum number of iterations the algorithm performs. Defaults to 5000.
#' @param lambda      A scalar hyperparameter trading off margin of the hyperplane versus average hinge
#'                    The default value is 1.
#' @return            The computed final parameter (weight) as a vector of length p + 1
linear_svm_learning_algorithm = function(Xinput, y_binary, MAX_ITER = 5000, lambda = 0.1){
  # Set n to be the length of Xinput
  # Set w to be an array of 0's the length of the columns of Xinput
  # Iterate MAX_ITER times
  # Iterate over the number of rows in XInput
  # Set x_i to be the i-eth row in Xinput
  # Set y_i to be the i-eth value in y_binary
  # Set sum_hinge_error to be the sum of the maximum
  # Calculate w vector
}
```

If you are enrolled in 342W the following is extra credit but if you're enrolled in 650, the following is required. Write the actual code. You may want to take a look at the `optimx` package. You can feel free to define another function (a "private" function) in this chunk if you wish. R has a way to create public and private functions, but I believe you need to create a package to do that (beyond the scope of this course).

```
#' This function implements the hinge-loss + maximum margin linear support vector machine algorithm of
#'
#' @param Xinput      The training data features as an n x p matrix.
#' @param y_binary    The training data responses as a vector of length n consisting of only 0's and 1'
#' @param MAX_ITER    The maximum number of iterations the algorithm performs. Defaults to 5000.
#' @param lambda      A scalar hyperparameter trading off margin of the hyperplane versus average hinge
#'                    The default value is 1.
#' @return            The computed final parameter (weight) as a vector of length p + 1
linear_svm_learning_algorithm = function(Xinput, y_binary, MAX_ITER = 5000, lambda = 0.1){
  n = nrow(Xinput)
  Xinput = as.matrix(cbind(rep(1, n), Xinput))
  w = rep(0, ncol(Xinput))
  for ( iter in 1 : MAX_ITER){
    for (i in 1 : nrow(Xinput)){
```

```
      x_i = Xinput[i, ]
      y_i = y_binary[i]
      sum_hinge_error = max(0, 0.5-(y_i-0.5)*(x_i * w - w))
      w = 1 / n * (sum_hinge_error) + lambda * norm(w) ^ 2
    }
  }
  w
}
```

If you wrote code (the extra credit), run your function using the defaults and plot it in brown vis-a-vis the previous model's line:

```
#y_binary = as.matrix(Xy_simple$response)
#X_simple_feature_matrix = as.matrix(Xy_simple)
#X_simple_feature_matrix$response = NULL

#svm_model_weights = linear_svm_learning_algorithm(X_simple_feature_matrix, y_binary)
#my_svm_line = geom_abline(
#    intercept = svm_model_weights[1] / svm_model_weights[3],#NOTE: negative sign removed from intercep
#    slope = -svm_model_weights[2] / svm_model_weights[3],
#    color = "brown")
#simple_viz_obj  + my_svm_line
```

Is this the same as what the `e1071` implementation returned? Why or why not?

TO-DO

We now move on to simple linear modeling using the ordinary least squares algorithm.

Let's quickly recreate the sample data set from practice lecture 7:

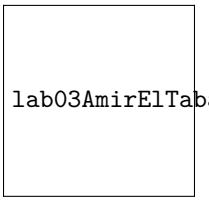
```
n = 20
x = runif(n)
beta_0 = 3
beta_1 = -2
```

Compute $h^*(x)$ as `h_star_x, then draw $\epsilon \sim N(0, 0.33^2)$ asepsilon‘, then compute .

```
h_star_x = beta_0 + beta_1 * x
epsilon = rnorm(n, 0, 0.33)
y = h_star_x + epsilon
```

Graph the data by running the following chunk:

```
pacman::p_load(ggplot2)
simple_df = data.frame(x = x, y = y)
simple_viz_obj = ggplot(simple_df, aes(x, y)) +
  geom_point(size = 2)
simple_viz_obj
```



lab03AmirElTabakh_files/figure-latex/unnamed-chunk-11-1.pdf

Does this make sense given the values of $beta_0$ and $beta_1$?

Write a function `my_simple_ols` that takes in a vector `x` and vector `y` and returns a list that contains the `b_0` (intercept), `b_1` (slope), `yhat` (the predictions), `e` (the residuals), `SSE`, `SST`, `MSE`, `RMSE` and `Rsq` (for the R-squared metric). Internally, you can only use the functions `sum` and `length` and other basic arithmetic operations. You should throw errors if the inputs are non-numeric or not the same length. You should also name the class of the return value `my_simple_ols_obj` by using the `class` function as a setter. No need to create ROxygen documentation here.

```
my_simple_ols = function(x, y){
  n = length(y)
  if (n != length(x)){
    stop("x and y need to be the same length.")
  }
  if(class(x) !='numeric' && class(x) !='integer'){
    stop("x needs to be numeric.")
  }
  if(class(y) !='numeric' && class(y) !='integer'){
    stop("y needs to be numeric.")
   }
  if(n<2){
    stop("n must be more than 2.")
  }
  x_bar = sum(x)/n
  y_bar = sum(y)/n
  b_1 = (sum(x*y)-n*x_bar*y_bar) / (sum(x^2)-n*x_bar^2)
  b_0 = y_bar - b_1*x_bar
  yhat = b_0 + b_1*x
  e = y - yhat
  SSE = sum(e^2)
  SST = sum((y - y_bar)^2)
  MSE = SSE / (n-2)
  RMSE = sqrt(MSE)
  Rsq = 1 - (SSE/SST)

  model = list(b_0=b_0, b_1 = b_1, yhat = yhat, e=e, SSE = SSE, SST = SST, MSE = MSE, RMSE = RMSE, Rsq =
  class(model) = "my_simple_ols_obj"

  model
}
```

Verify your computations are correct for the vectors `x` and `y` from the first chunk using the `lm` function in R:

```
lm_mod = lm(y~x)
my_simple_ols_mod = my_simple_ols(x, y)
#my_simple_ols_mod
#run the tests to ensure the function is up to spec
pacman::p_load(testthat)
expect_equal(my_simple_ols_mod$b_0, as.numeric(coef(lm_mod)[1]), tol = 1e-4)
expect_equal(my_simple_ols_mod$b_1, as.numeric(coef(lm_mod)[2]), tol = 1e-4)
expect_equal(my_simple_ols_mod$RMSE, summary(lm_mod)$sigma, tol = 1e-4)
expect_equal(my_simple_ols_mod$Rsq, summary(lm_mod)$r.squared, tol = 1e-4)
```

Verify that the average of the residuals is 0 using the `expect_equal`. Hint: use the syntax above.

```
#average function can be close to 0, not 0.
expect_equal(mean(my_simple_ols_mod$e), 0)
mean(my_simple_ols_mod$e)
```

```
## [1] -1.221164e-16
```

Create the X matrix for this data example. Make sure it has the correct dimension.

```
X = cbind(1, x)
X
```

```
##                   x
##  [1,] 1 0.49697304
##  [2,] 1 0.26172960
##  [3,] 1 0.79993297
##  [4,] 1 0.25117131
##  [5,] 1 0.14434941
##  [6,] 1 0.92838371
##  [7,] 1 0.70017575
##  [8,] 1 0.58100840
##  [9,] 1 0.55644748
## [10,] 1 0.03782235
## [11,] 1 0.65564673
## [12,] 1 0.33205484
## [13,] 1 0.71644702
## [14,] 1 0.61952602
## [15,] 1 0.13167843
## [16,] 1 0.56970599
## [17,] 1 0.90234184
## [18,] 1 0.55781643
## [19,] 1 0.87135187
## [20,] 1 0.33617013
```

Use the `model.matrix` function to compute the matrix `X` and verify it is the same as your manual construction.

```
model.matrix(~x)
```

```
##    (Intercept)          x
## 1            1 0.49697304
## 2            1 0.26172960
## 3            1 0.79993297
## 4            1 0.25117131
## 5            1 0.14434941
## 6            1 0.92838371
## 7            1 0.70017575
## 8            1 0.58100840
## 9            1 0.55644748
## 10           1 0.03782235
## 11           1 0.65564673
## 12           1 0.33205484
## 13           1 0.71644702
## 14           1 0.61952602
## 15           1 0.13167843
## 16           1 0.56970599
## 17           1 0.90234184
## 18           1 0.55781643
## 19           1 0.87135187
## 20           1 0.33617013
## attr(,"assign")
## [1] 0 1
```

Create a prediction method `g` that takes in a vector `x_star` and `my_simple_ols_obj`, an object of type `my_simple_ols_obj` and predicts y values for each entry in `x_star`.

```
g = function(my_simple_ols_obj, x_star){
  y_star = my_simple_ols_obj$b_0 + my_simple_ols_obj$b_1*x_star
}
```

Use this function to verify that when predicting for the average x, you get the average y.

```
expect_equal(g(my_simple_ols_mod, mean(x)), mean(y))
```

In class we spoke about error due to ignorance, misspecification error and estimation error. Show that as n grows, estimation error shrinks. Let us define an error metric that is the difference between b_0 and b_1 and β_0 and β_1. How about $h = ||b - \beta||^2$ where the quantities are now the vectors of size two. Show as n increases, this shrinks.

```
beta_0 = 3
beta_1 = -2
beta = c(beta_0, beta_1)
ns = 10^(1:8)
error_in_b = array(NA, length(ns))
for (i in 1 : length(ns)) {
  n = ns[i]
  x = runif(n)
  h_star_x = beta_0 + beta_1 * x
  epsilon = rnorm(n, mean = 0, sd = 0.33)
  y = h_star_x + epsilon

  mod = my_simple_ols(x, y)
  b = c(mod$b_0, mod$b_1)

  error_in_b[i] = sum((beta - b)^2)
}
log(error_in_b, 10)
```

```
## [1] -0.5896984 -1.7906521 -2.7629037 -5.0075307 -4.4964269 -6.4747795 -8.0734644
## [8] -9.0413780
```

We are now going to repeat one of the first linear model building exercises in history — that of Sir Francis Galton in 1886. First load up package `HistData`.

```
pacman::p_load(HistData)
```

In it, there is a dataset called `Galton`. Load it up.

```
data(Galton)
```

You now should have a data frame in your workspace called `Galton`. Summarize this data frame and write a few sentences about what you see. Make sure you report n, p and a bit about what the columns represent and how the data was measured. See the help file `?Galton`. p is 1 and n is 928 the number of observations

```
pacman::p_load(skimr)
skim(Galton)
```

Table 1: Data summary

Name	Galton
Number of rows	928
Number of columns	2

Table 1: Data summary

Column type frequency:	
numeric	2
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
parent	0	1	68.31	1.79	64.0	67.5	68.5	69.5	73.0	
child	0	1	68.09	2.52	61.7	66.2	68.2	70.2	73.7	

TO-DO n is 928, and p would be 1. That is, there are 928 rows, and one independent variable being height in inches. There are two values in target column, child and parent. The median of both parents and children seem to be very close, 68.5 and 68.2 respectively. The mean of both groups are also very close, 68.3 is the mean height for parents, and 68.1 is the mean height for children. This makes sense, as there is no reason why there would be a significant change in height between two generations of children. Find the average height (include both parents and children in this computation).

```
avg_height = mean(c(Galton$parent, Galton$child))
```

If you were predicting child height from parent hight and you were to use the null model, what would the RMSE be of this model be?

```
n = nrow(Galton)
SST = sum((Galton$child - mean(Galton$child))^2)
rmse = sqrt(SST/(n-1))
```

Note that in Math 241 you learned that the sample average is an estimate of the "mean", the population expected value of height. We will call the average the "mean" going forward since it is probably correct to the nearest tenth of an inch with this amount of data.

Run a linear model attempting to explain the childrens' height using the parents' height. Use `lm` and use the R formula notation. Compute and report b_0, b_1, RMSE and R^2.

```
mod = lm(Galton$child~Galton$parent, Galton)
b_0 = coef(mod)[1]
b_1 = coef(mod)[2]
b_0
```

```
## (Intercept)
##    23.94153
```

```
b_1
```

```
## Galton$parent
##     0.6462906
```

```
summary(mod)$r.squared
```

```
## [1] 0.2104629
```

```
summary(mod)$sigma
```

```
## [1] 2.238547
```

Interpret all four quantities: b_0, b_1, RMSE and R^2. Use the correct units of these metrics in your answer.

The y-intercept is 23.9 inches, which makes little sense because that implies that for parents with an average height of 0 inches, then the average height of their children will be 23.9. Nonetheless, this y-intercept is required for this model. The slope of this line is 0.65 inches, that is, for every one inch increase in average parent height, we see a 0.65 inch increase in child height. The RMSE is a standard way to measure the error of a model in predicting quantitative data. The RMSE of this dataset is 2.23*2, meaning given a parent height, the model can predict the height of the child plus or minus 2.24. The R^2 metric summarizes the models ability to fit the data. An R^2 of 0.21 indicates that only 21% of the variance in the dataset is explained by the model.

How good is this model? How well does it predict? Discuss.

The R^2 of the model accounts for 21% of the variance. Furthermore, according to the RMSE, the model can predict the value of a child given the height of the parent plus or minus 2.23*2. I believe this model is limited in how it can predict because the graph is linear.

It is reasonable to assume that parents and their children have the same height? Explain why this is reasonable using basic biology and common sense.

There is no reason for there to be a significant height difference between a parent and their children. Furthermore, the mean and median of children and parents are very similar.

If they were to have the same height and any differences were just random noise with expectation 0, what would the values of β_0 and β_1 be?

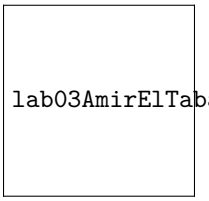
b_0 would be 0 and b_1 would be 1.

Let's plot (a) the data in $\mathbb{D}$ as black dots, (b) your least squares line defined by b_0 and b_1 in blue, (c) the theoretical line β_0 and β_1 if the parent-child height equality held in red and (d) the mean height in green.

```
pacman::p_load(ggplot2)
ggplot(Galton, aes(x = parent, y = child)) +
  geom_point() +
  geom_jitter() +
  geom_abline(intercept = b_0, slope = b_1, color = "blue", size = 1) +
  geom_abline(intercept = 0, slope = 1, color = "red", size = 1) +
  geom_abline(intercept = avg_height, slope = 0, color = "darkgreen", size = 1) +
  xlim(63.5, 72.5) +
  ylim(63.5, 72.5) +
  coord_equal(ratio = 1)
```

```
## Warning: Removed 76 rows containing missing values (geom_point).
```

```
## Warning: Removed 83 rows containing missing values (geom_point).
```



Fill in the following sentence:

Children of short parents became taller on average and children of tall parents became shorter on average.

Why did Galton call it "Regression towards mediocrity in hereditary stature" which was later shortened to "regression to the mean"?

Because of the relationship between average children heights and average parent heights is not what we expect. Child heights seem to 'regress' from the average height of parents. This is to say that the average height of children is not exactly the average height of their parents, the average height of children imperfectly represents that of their parents.

Why should this effect be real?

The evolution of mankind is not something that takes place over the course of a generation. This is basic biology.

You now have unlocked the mystery. Why is it that when modeling with y continuous, everyone calls it "regression"? Write a better, more descriptive and appropriate name for building predictive models with y continuous.

People call it a regression model because although we expect such a relationship to be nearly 1:1, we see that the average height of children seem to regress from the average height of parents. Linear expectancy.

You can now clear the workspace. Create a dataset $\mathbb{D}$ which we call `Xy` such that the linear model as R^2 about 50% and RMSE approximately 1.

```
x = 1:5
y = x^100
Xy = data.frame(x = x, y = y)

mod = lm(x~y, Xy)
summary(mod)$r.squared
```

```
## [1] 0.5
```

```
summary(mod)$sigma
```

```
## [1] 1.290994
```

Create a dataset $\mathbb{D}$ which we call `Xy` such that the linear model as R^2 about 0% but x, y are clearly associated.

```
x = c(100, 100, 100, 100)
y = c(1, 1, 1, 1)
Xy = data.frame(x = x, y = y)

mod = lm(x~y, Xy)
summary(mod)$r.squared
```

```
## Warning in summary.lm(mod): essentially perfect fit: summary may be unreliable
```

```
## [1] 0
```

```
summary(mod)$sigma
```

```
## Warning in summary.lm(mod): essentially perfect fit: summary may be unreliable
```

```
## [1] 0
```

Extra credit: create a dataset $\mathbb{D}$ and a model that can give you R^2 arbitrarily close to 1 i.e. approximately 1 - epsilon but RMSE arbitrarily high i.e. approximately M.

```
epsilon = 0.01
M = 1000

#Trick question, they're inversely related, therefore this is not possible. RMSE is the sum of errors s
```

Write a function `my_ols` that takes in `X`, a matrix with with p columns representing the feature measurements for each of the n units, a vector of n responses `y` and returns a list that contains the `b`, the $p+1$-sized column

vector of OLS coefficients, `yhat` (the vector of n predictions), `e` (the vector of n residuals), `df` for degrees of freedom of the model, `SSE`, `SST`, `MSE`, `RMSE` and `Rsq` (for the R-squared metric). Internally, you cannot use `lm` or any other package; it must be done manually. You should throw errors if the inputs are non-numeric or not the same length. Or if `X` is not otherwise suitable. You should also name the class of the return value `my_ols` by using the `class` function as a setter. No need to create ROxygen documentation here.

```
my_ols = function(X, y){
  n = length(y)
  if (!is.numeric(X) && !is.integer(X)) {
    stop("X is not numeric")
  }
  X = cbind(rep(1, n), X)
  p = ncol(X)
  df = ncol(X)
  if (n != nrow(X)){
    stop("X rows and length of y need to be the same length.")
  }
  if(class(y) !='numeric' && class(y) !='integer'){
    stop("y needs to be numeric.")
   }
  if(n<=ncol(X)+1){
    stop("n must be more than 2.")
  }

  y_bar = sum(y)/n

  b = solve(t(X) %*% X) %*% t(X) %*% y
  yhat = X %*% b

  e = y - yhat
  SSE = t(e) %*% e
  SST = t(y - y_bar) %*% (y - y_bar)
  MSE = SSE / (n-(p+1))
  RMSE = sqrt(MSE)
  Rsq = 1 - (SSE/SST)

  model = list(b=b, yhat = yhat, df = df, e=e, SSE = SSE, SST = SST, MSE = MSE, RMSE = RMSE, Rsq = Rsq,
  class(model) = "my_ols_obj"

  model
}
```

Verify that the OLS coefficients for the `Type` of cars in the cars dataset gives you the same results as we did in class (i.e. the ybar's within group).

```
cars = MASS::Cars93
mod = lm(Price~Type, data=cars)
x = my_ols(as.numeric(data.matrix(data.frame((cars$Type)))), cars$Price)
```

Create a prediction method `g` that takes in a vector `x_star` and the dataset $\mathbb{D}$ i.e. `X` and `y` and returns the OLS predictions. Let `X` be a matrix with with p columns representing the feature measurements for each of the n units

```
g = function(x_star, X, y){
  b = my_ols(X,y)$b
  x_star = c(1,x_star)
```

```
  x_star %*% b
}
X = model.matrix( ~Type,cars)[, 2:6]

g_func_val = g(X[1,], X, cars$Price)
manual = t(c(1,X[1,])) %*% my_ols(X, cars$Price)$b
predict = predict(mod, cars[1,])

g_func_val
```

```
##          [,1]
## [1,] 10.16667
```

```
manual
```

```
##          [,1]
## [1,] 10.16667
```

```
predict
```

```
##        1
## 10.16667
```