

Support Vector Machines

SYS 6018 | Spring 2023

svm.pdf

Contents

1	Support Vector Machines (SVM) Introduction	2
1.1	Required R Packages	2
1.2	Example	2
1.3	SVM as Loss + Penalty	3
1.4	SVM vs. Logistic Regression	4
1.5	Example: Compare Linear Classifiers	5
1.6	Three Representations of the Optimization Problem	6
1.7	Linear SVM	6
2	Kernels and Non-linear SVM	7
2.1	Basis Expansion	7
2.2	Alternative (Dual) Formulation	8
2.3	SVM in R	8
2.4	Kernels	9
2.5	Unbalanced Data	12
3	Appendix: R Code	13

1 Support Vector Machines (SVM) Introduction

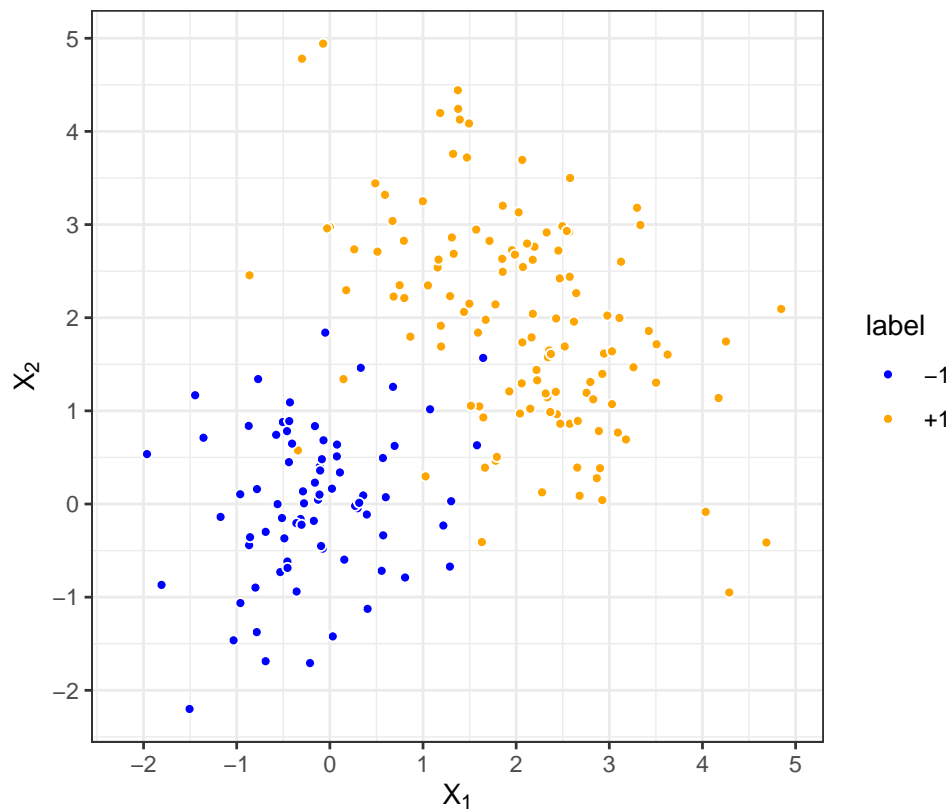
1.1 Required R Packages

We will be using the R packages of:

- `tidyverse` for data manipulation and visualization
- `e1071` for the `svm()` functions

1.2 Example

Goal: find *best* line(s)/curve(s) to separate the two classes.



1.3 SVM as Loss + Penalty

Training Data: $\{(\tilde{y}_i, \mathbf{x}_i)\}_{i=1}^n$

- $\tilde{y}_i \in \{-1, +1\}$
- $\mathbf{x}_i^\top = [x_{i1}, x_{i2}, \dots, x_{in}] \in \mathbb{R}^p$

Predictor Function: $f(x)$

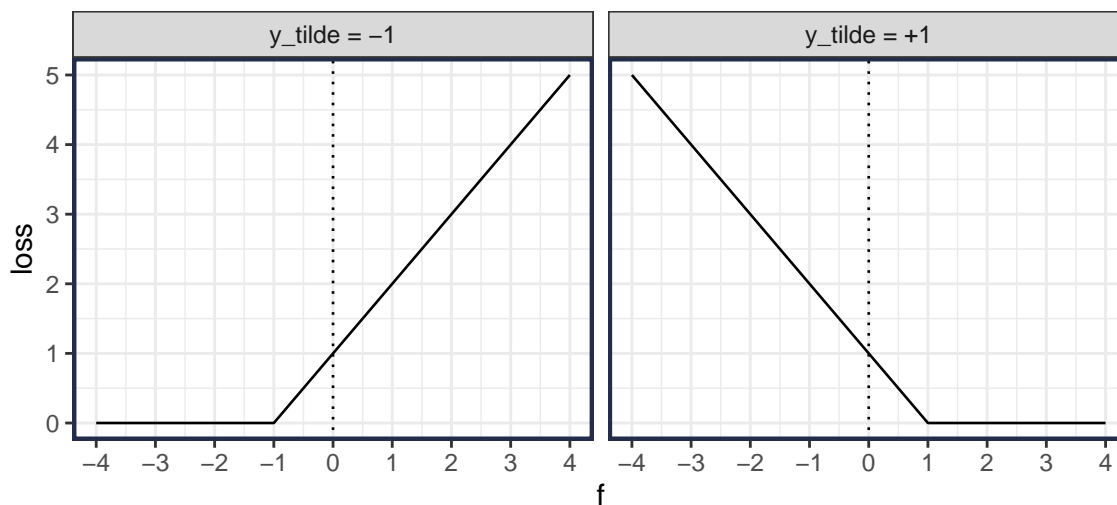
- Linear: $f(\mathbf{x}; \beta) = \beta_0 + \sum_{j=1}^p x_j \beta_j$
- Basis Expansion: $f(\mathbf{x}; \beta) = \beta_0 + \sum_{j=1}^d h_j(\mathbf{x}) \beta_j$
 - $h_j(\mathbf{x})$ transforms the raw x vector
 - Polynomial example: $h_1(\mathbf{x}) = x_1$, $h_2(\mathbf{x}) = x_1^2$, $h_3(\mathbf{x}) = x_2$, $h_4(\mathbf{x}) = x_2^2$, $h_5(\mathbf{x}) = x_1 x_2$
 - This is the connection to *kernels* that we will discuss later
- Let $f_i = f(\mathbf{x}_i)$

Classification:

- If $\hat{f}_i \geq 0$, then label as class +1
- If $\hat{f}_i < 0$, then label as class -1

Loss Function: Hinge Loss

$$L(\tilde{y}_i, f_i) = \max\{0, 1 - \tilde{y}_i f_i\}$$



Penalty Function: Ridge Penalty

$$P_\lambda(\beta) = \frac{\lambda}{2} \sum_{j=1}^d \beta_j^2$$

$$= \lambda \|\beta\|^2 / 2$$

- This should have you thinking that the \mathbf{x} 's should be scaled!

Summary of SVM

1. Estimate *model parameters* β

$$\begin{aligned}\hat{\beta}_\lambda &= \arg \min_{\beta} \left\{ \sum_{i=1}^n \max\{0, 1 - \tilde{y}_i f_i(\beta)\} + \lambda \|\beta\|^2 / 2 \right\} \\ &= \arg \min_{\beta} \{ \text{Hinge Loss}(\beta) + \lambda \text{Penalty}(\beta) \}\end{aligned}$$

2. Label Class +1 if $\hat{f}_i \geq 0$, else label Class -1
 - This implicitly assumes a 0-1 loss (or equal cost FP and FN)
 - More generally, use threshold t that considers the costs of FP and FN
3. **Tuning Parameters:** besides the λ , SVM will also have tuning parameters related to the kernels (more to come on this).

1.4 SVM vs. Logistic Regression

The *linear* SVM is actually very similar to Logistic Regression (with Ridge Penalty).

Let $f_i(\beta) = \beta_0 + \sum_{j=1}^p x_j \beta_j$ be the log-odds.

- We also referred to this as $\gamma(\mathbf{x}_i)$ in previous notes.

1. Estimate *model parameters* β

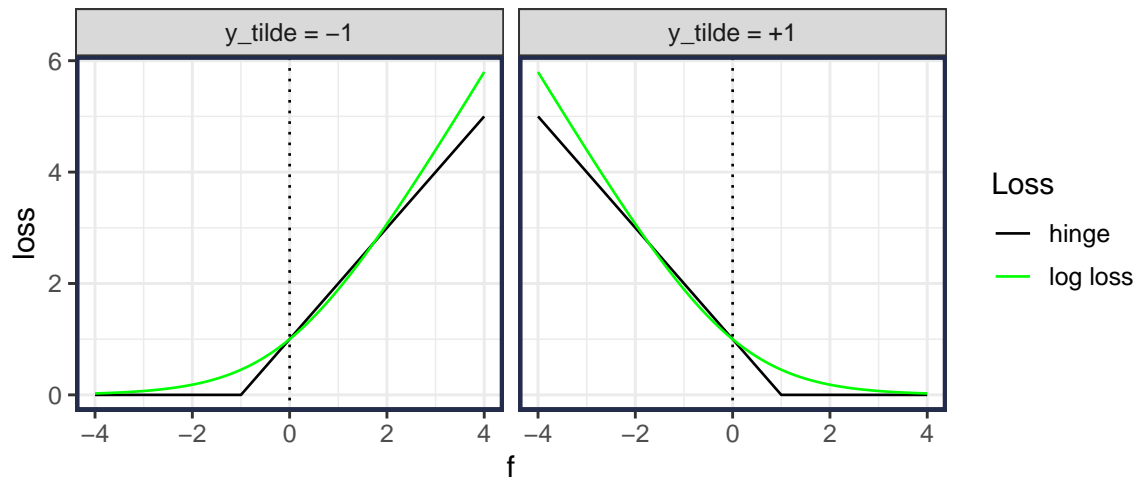
- This assumes standardized \mathbf{x} 's

$$\begin{aligned}\hat{\beta}_\lambda &= \arg \min_{\beta} \left\{ \sum_{i=1}^n \log(1 + \exp(-\tilde{y}_i f_i(\beta))) + \lambda \|\beta\|^2 / 2 \right\} \\ &= \arg \min_{\beta} \{ \text{Log Loss}(\beta) + \lambda \text{Penalty}(\beta) \}\end{aligned}$$

2. Label Class +1 if $\hat{f}_i \geq t$, else label Class -1
 - For some threshold t that considers the costs of FP and FN
 - If $\text{Cost}(\text{FP}) = \text{Cost}(\text{FN})$, the set $t = 0$, which is equivalent to $p(x) = 1/2$

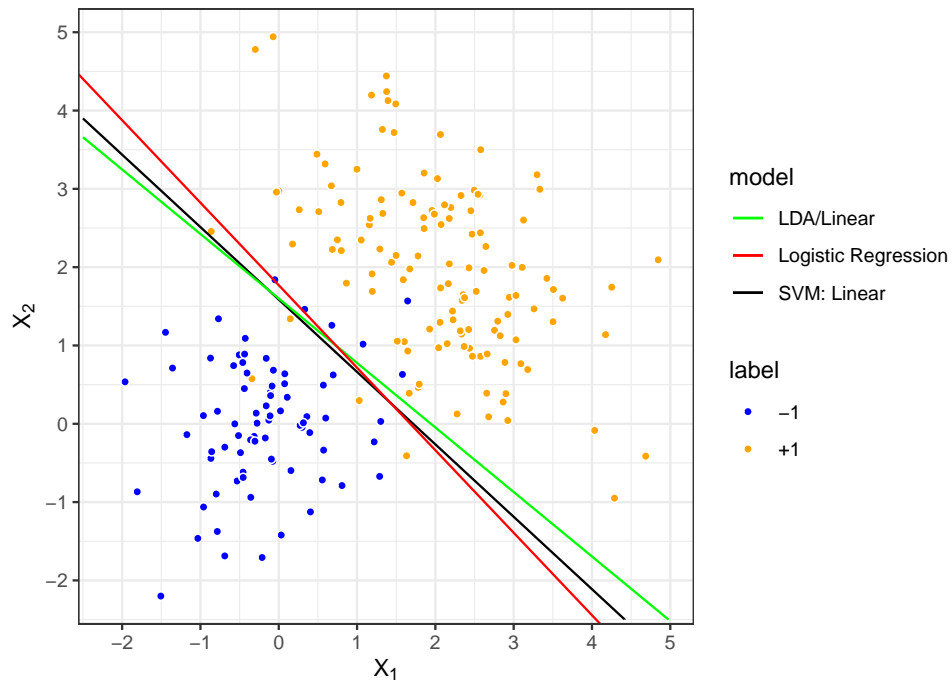
Details of the Logistic Regression Loss Function

Note



The Log-Loss has been scaled so it equals the Hinge Loss at $f=0$.

1.5 Example: Compare Linear Classifiers



1.6 Three Representations of the Optimization Problem

1. Penalized optimization

$$\begin{aligned}\hat{\beta} &= \arg \min_{\beta} \{ \text{Loss}(\beta) + \lambda \text{Penalty}(\beta) \} \\ &= \arg \min_{\beta} \{ C \text{Loss}(\beta) + \text{Penalty}(\beta) \}\end{aligned}$$

- where $C = \frac{1}{\lambda} > 0$ is an alternative strength of penalty
- In SVM, C is referred to as the *cost*

2. Constraint on Penalty

$$\begin{aligned}\hat{\beta} &= \arg \min_{\beta} \text{Loss}(\beta) \quad \text{subject to } \text{Penalty}(\beta) \leq t \\ &= \arg \min_{\beta: \text{Penalty}(\beta) \leq t} \text{Loss}(\beta)\end{aligned}$$

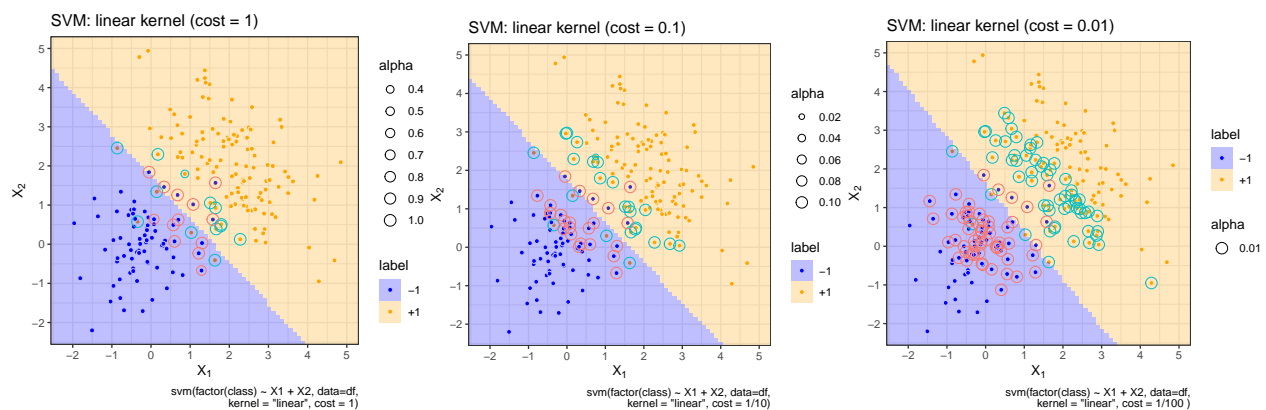
3. Constraint on Loss

$$\begin{aligned}\hat{\beta} &= \arg \min_{\beta} \text{Penalty}(\beta) \quad \text{subject to } \text{Loss}(\beta) \leq M \\ &= \arg \min_{\beta: \text{Loss}(\beta) \leq M} \text{Penalty}(\beta)\end{aligned}$$

Note

There is a one-to-one relationship between all tuning parameters: λ , C , t , M .

1.7 Linear SVM



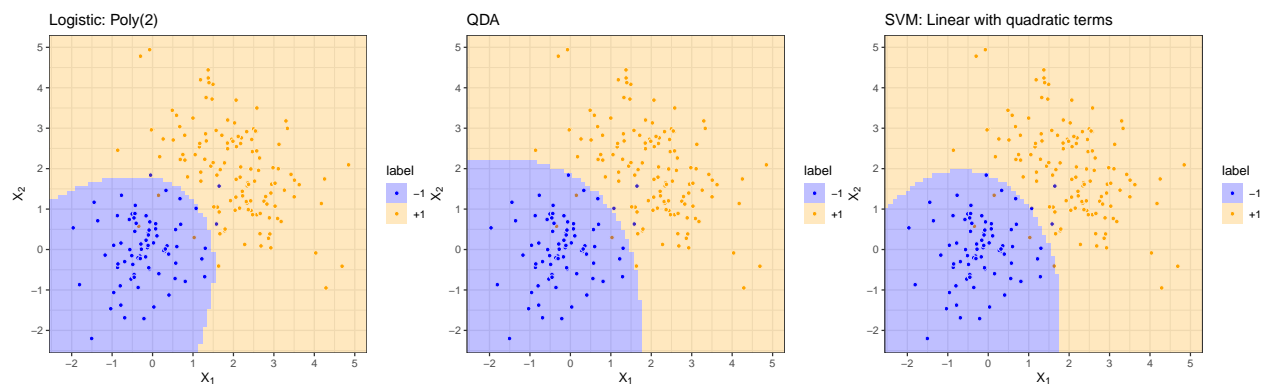
2 Kernels and Non-linear SVM

2.1 Basis Expansion

- Linear: $f(\mathbf{x}; \beta) = \beta_0 + \sum_{j=1}^p x_j \beta_j$
- Basis Expansion: $f(\mathbf{x}; \beta) = \beta_0 + \sum_{j=1}^d h_j(\mathbf{x}) \beta_j$
 - $h_j(\mathbf{x})$ transforms the raw \mathbf{x} vector

2.1.1 Polynomial Expansion: Quadratic Terms

- Polynomial example: $h_1(\mathbf{x}) = x_1$, $h_2(\mathbf{x}) = x_1^2$, $h_3(\mathbf{x}) = x_2$, $h_4(\mathbf{x}) = x_2^2$, $h_5(\mathbf{x}) = x_1 x_2$
- $f(\mathbf{x}; \beta) = \beta_0 + \sum_{j=1}^5 h_j(\mathbf{x}) \beta_j$



2.2 Alternative (Dual) Formulation

It turns out that the SVM solution for the model coefficients β can be written as

$$\hat{\beta}_j = \sum_{i=1}^n \hat{\alpha}_i \tilde{y}_i h_j(\mathbf{x}_i)$$

- See ESL Eq. 12.17 if you are interested in the details.
- In this reformulation, the $\{\hat{\alpha}_i\}_{i=1}^n$ become the model parameters.
 - There is one model parameter for each observation!
 - $0 \leq \alpha_i \leq C$ (or $0 \leq \alpha_i \leq \frac{1}{\lambda}$)
 - But SVM will force most values to 0.
 - The observations with $\alpha_i > 0$ are called the *support vectors*
 - * So the entire SVM model is a function of the support vectors only!
 - * The support vectors are the observations on the wrong side of the margin.
- The decision function $\hat{f}(\mathbf{x})$ can be re-written

$$\begin{aligned} \hat{f}(\mathbf{x}) &= f(\mathbf{x}; \hat{\beta}) \\ &= \hat{\beta}_0 + \sum_{j=1}^d h_j(\mathbf{x}) \hat{\beta}_j \\ &= \hat{\beta}_0 + \sum_{j=1}^d h_j(\mathbf{x}) \left[\sum_{i=1}^n \hat{\alpha}_i \tilde{y}_i h_j(\mathbf{x}_i) \right] \\ &= \hat{\beta}_0 + \sum_{i=1}^n \hat{\alpha}_i \tilde{y}_i \left[\sum_{j=1}^d h_j(\mathbf{x}) h_j(\mathbf{x}_i) \right] \\ &= \hat{\beta}_0 + \sum_{i=1}^n \hat{\alpha}_i \tilde{y}_i K(\mathbf{x}, \mathbf{x}_i) \end{aligned}$$

where

$$K(\mathbf{x}, \mathbf{x}_i) = \sum_{j=1}^d h_j(\mathbf{x}) h_j(\mathbf{x}_i) = \langle h(\mathbf{x}), h(\mathbf{x}_i) \rangle$$

is called a **kernel** and measures the inner product, or *similarity* between x and x_i (observation i).

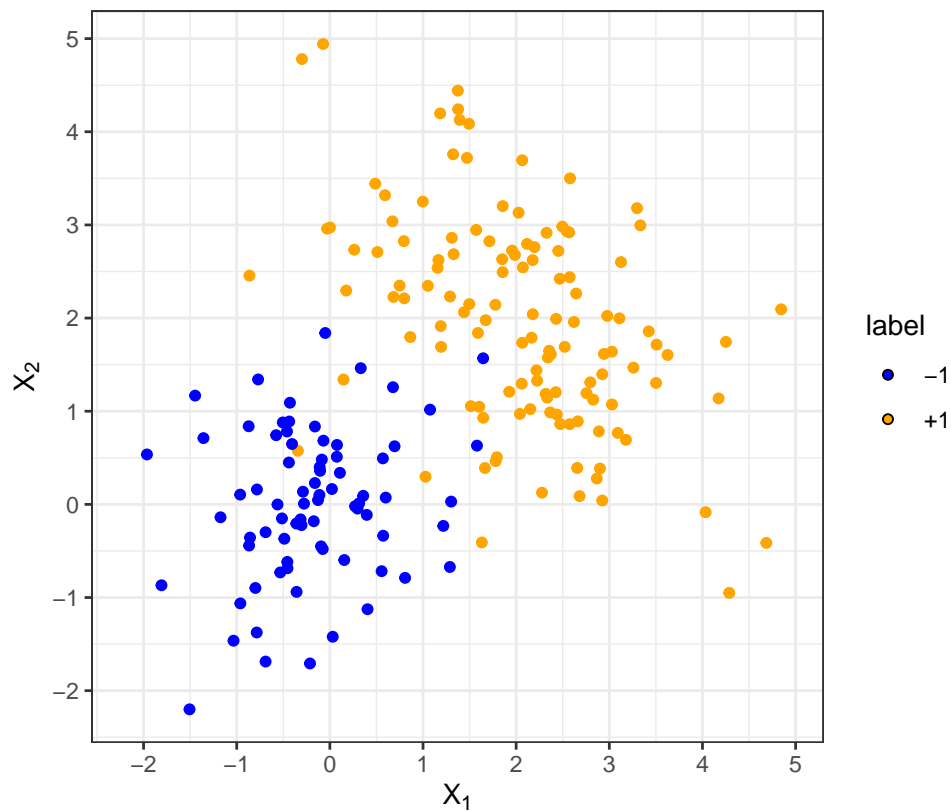
2.3 SVM in R

- The `svm()` function from the `e1071` package can implement SVM.
 - There is also a helpful `tune.svm()` to help you select the tuning parameters.
- The ISLR Lab in Section 9.6 has example R code.

2.4 Kernels

To help illustrate the difference between different kernels, let's look at slightly different data that won't be easy to classify using linear classifiers.

```
#> Error in rmvnorm(n1, mean = mu1, sigma = sigma1): unused argument (mean = mu1)
#> Error in rmvnorm(n2[1], mean = mu2, sigma = sigma2): unused argument (mean = mu2)
#> Error in rmvnorm(n2[2], mean = mu2.B, sigma = sigma2): unused argument (mean = mu2.B)
```

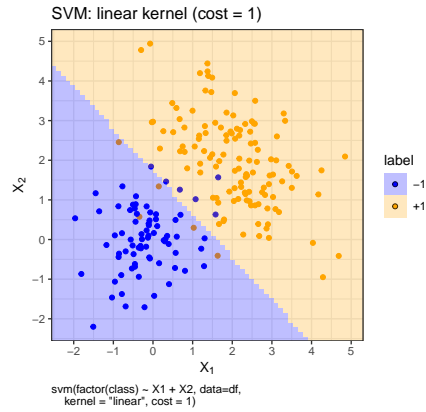


Three popular kernels (but there are many more) are the *linear*, *polynomial*, and *radial basis*

2.4.1 Linear Kernel

The *linear kernel* is

$$K(\mathbf{x}, \mathbf{u}) = \sum_{j=1}^p \mathbf{x}_j \mathbf{u}_j = \langle \mathbf{x}, \mathbf{u} \rangle$$



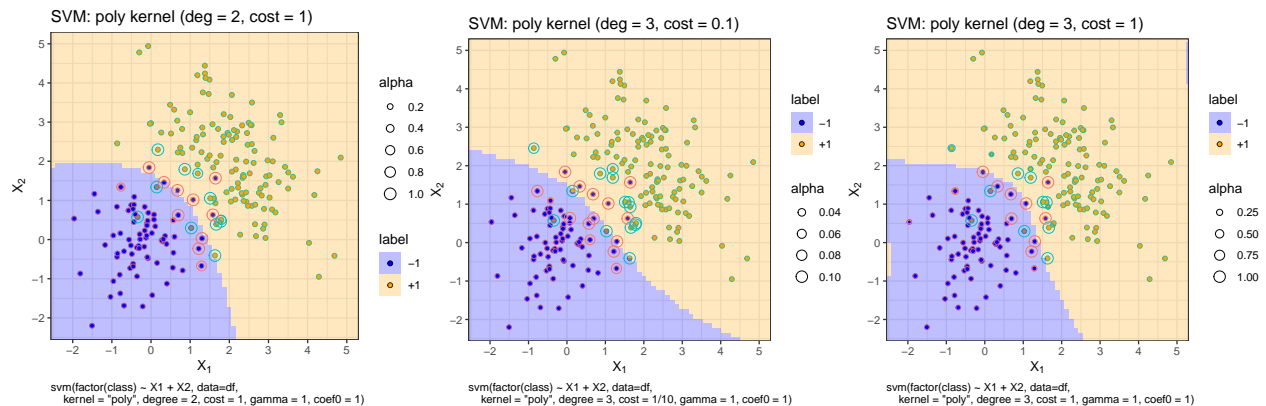
$$\begin{aligned}\hat{f}_{\text{linear}}(\mathbf{u}) &= \hat{\beta}_0 + \sum_{i=1}^n \hat{\alpha}_i \tilde{y}_i K(\mathbf{x}_i, \mathbf{u}) \\ &= \hat{\beta}_0 + \sum_{i=1}^n \hat{\alpha}_i \tilde{y}_i \left(\sum_{j=1}^p \mathbf{x}_{ij} \mathbf{u}_j \right)\end{aligned}$$

2.4.2 Polynomial Kernel

The *polynomial kernel* of degree `deg` is

$$K(\mathbf{x}, \mathbf{u}) = \left(1 + \sum_{j=1}^p \mathbf{x}_j \mathbf{u}_j \right)^{\text{deg}}$$

Note: in R, the 'svm()' function from 'e1071' package includes two other tuning parameters (`gamma` and `coef0`)



$$\begin{aligned}\hat{f}_{\text{poly}}(\mathbf{u}) &= \hat{\beta}_0 + \sum_{i=1}^n \hat{\alpha}_i y_i K(\mathbf{x}_i, \mathbf{u}) \\ &= \hat{\beta}_0 + \sum_{i=1}^n \hat{\alpha}_i y_i \left(1 + \sum_{j=1}^p \mathbf{x}_{ij} \mathbf{u}_j \right)^{\text{deg}}\end{aligned}$$

2.4.3 Radial Basis Kernel

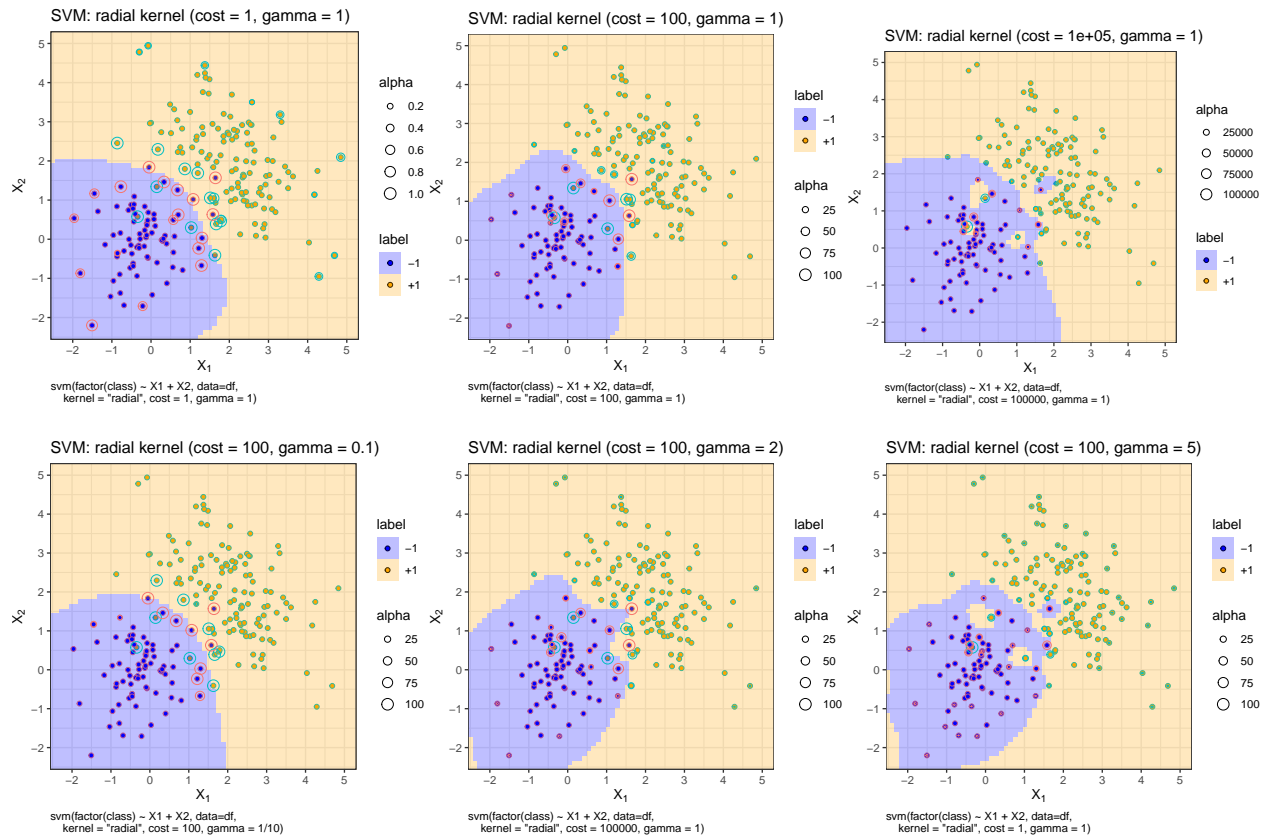
The *Radial Basis Kernel* with parameter γ (`gamma`) is

$$K(\mathbf{x}, \mathbf{u}) = \exp \left(-\gamma \sum_{j=1}^p (\mathbf{x}_j - \mathbf{u}_j)^2 \right)$$

$$= \exp \left(-\gamma \text{dist}^2(\mathbf{x}, \mathbf{u}) \right)$$

where $\text{dist}^2(\mathbf{x}, \mathbf{u})$ is the *squared Euclidean distance* between \mathbf{x} and \mathbf{u} .

- The Radial Basis kernel is large for test observations close to training observations.
- Notice that both `cost` and `gamma` are influential tuning parameters.



$$\hat{f}_{\text{radial}}(\mathbf{u}) = \hat{\beta}_0 + \sum_{i=1}^n \hat{\alpha}_i \tilde{y}_i K(\mathbf{x}_i, \mathbf{u})$$

$$= \hat{\beta}_0 + \sum_{i=1}^n \hat{\alpha}_i \tilde{y}_i \exp \left(-\gamma \text{dist}^2(\mathbf{x}_i, \mathbf{u}) \right)$$

2.5 Unbalanced Data

Three primary approaches:

1. Use weights corresponding to prior class probabilities
 - See the `class.weights` argument in `?svm`
 - Ensure the resulting score is what you want for unadjusted prediction.

$$\hat{\beta}_{\lambda} = \arg \min_{\beta} \left\{ \sum_{i=1}^n w_i \max\{0, 1 - \tilde{y}_i f_i(\beta)\} + \lambda \|\beta\|^2 / 2 \right\}$$

2. Use a threshold on the score
 - E.g., choose label = +1 if: $\hat{f}(\mathbf{x}) > t$
 - The \hat{f} are the `decision.values` output of `predict.svm`
 - Its an attribute of the returned object
3. Sampling
 - under or over sample training data from one class to balance the label distribution.
 - The SVM (hinge) objective function is optimized at $\text{sign} [\text{Pr}(Y = +1|X = x) - 1/2]$.
 - Ensure the resulting score is what you want for unadjusted prediction.

3 Appendix: R Code

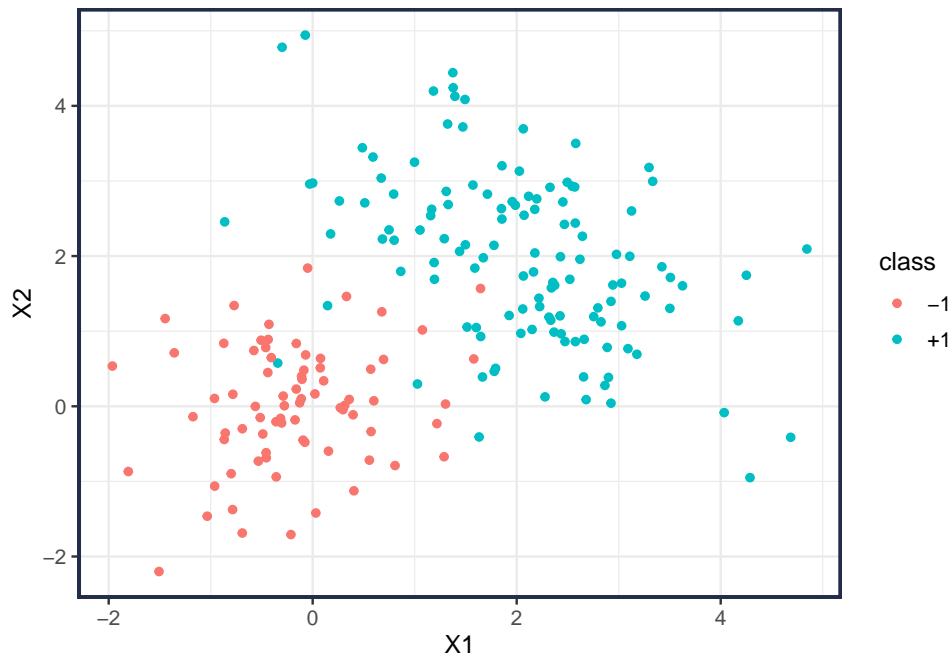
```
#-----#
# Create Data
#-----#
library(mvtnorm)

prior = c(.60, .40)
mu1 = c(2, 2)
mu2 = c(0, 0)
mu2.B = c(4, 4)
sigma1 = .5*matrix(c(2,-1, -1, 2), nrow=2)
sigma2 = .5*matrix(c(1,0,0,1), nrow=2)

set.seed(2020)
n = 200
n1 = rbinom(1, size=n, prob=prior[1])
n2 = n-n1
n2 = round(c(n2/2, n2/2))
X1 = rmvnorm(n1, mean=mu1, sigma = sigma1)
#> Error in rmvnorm(n1, mean = mu1, sigma = sigma1): unused argument (mean = mu1)
X2 = rmvnorm(n2[1], mean=mu2, sigma = sigma2)
#> Error in rmvnorm(n2[1], mean = mu2, sigma = sigma2): unused argument (mean = mu2)
X2 = rbind(X2, rmvnorm(n2[2], mean=mu2.B, sigma = sigma2))
#> Error in rmvnorm(n2[2], mean = mu2.B, sigma = sigma2): unused argument (mean = mu2.B)
labels = c("+1", "-1")

data_mix = bind_rows(
  !!labels[1] := as_tibble(X1, .name_repair = ~str_c("X", seq_along(.))),
  !!labels[2] := as_tibble(X2, .name_repair = ~str_c("X", seq_along(.))),
  .id = "class"
)

data_mix %>%
  ggplot(aes(X1, X2, color = class)) +
  geom_point()
```



```
#-----#
# Create Cross-Validation Folds
#-----#
set.seed(2022)
n.folds = 10
fold = sample(rep(1:n.folds, length=nrow(data_mix)))

# using tidymodels
library(rsample)
set.seed(2022)
cv_folds = rsample::vfold_cv(data_mix, v = 10)

#-----#
# Polynomial SVM Example
#-----#
library(e1071)
fit = svm(factor(class) ~ X1 + X2,
  data = data_mix[fold != 1, ], # hold out fold 1
  #: tuning parameters
  kernel = "poly",
  degree = 2,
  cost = 1
)

# Notes:
# - svm() requires that the outcome variable be a "factor" for classification
#   problems.
# - the polynomial kernel only has cost and degree parameters

#: basic model summary
summary(fit)
#>
#> Call:
#> svm(formula = factor(class) ~ X1 + X2, data = data_mix[fold != 1,
#>   ], kernel = "poly", degree = 2, cost = 1)
#>
#>
```

```

#> Parameters:
#>   SVM-Type:  C-classification
#>   SVM-Kernel: polynomial
#>     cost:  1
#>   degree:  2
#>   coef.0:  0
#>
#> Number of Support Vectors:  108
#>
#>   ( 54 54 )
#>
#>
#> Number of Classes:  2
#>
#> Levels:
#>   -1 +1

# : predictions
pred = predict(fit, data_mix[fold == 1,], decision.values = TRUE)

eval_data = tibble(
  outcome = data_mix$class[fold == 1],
  pred_hard = c(pred),
  pred_soft = as.numeric(attr(pred, "decision.values"))
)

# Notes:
# - using the `decision.values = TRUE` argument to get the scores. But set
#   to default of FALSE to just get the hard classifications
# - the scores can be used to create rank or threshold-based metrics like ROC
#   curves
# - There is also a way to get probability estimates by setting
#   svm(..., probability = TRUE) and predict(..., probability = TRUE).
#   This attempts to convert scores to probabilities

# : evaluation
library(yardstick)
levs = c("+1", "-1") # set outcome of interest at first level

eval_data %>%
  yardstick::roc_curve(truth = factor(outcome, levels=levs), estimate = pred_soft)
#> # A tibble: 22 x 3
#>   .threshold specificity sensitivity
#>   <dbl>         <dbl>         <dbl>
#> 1    -Inf             0             1
#> 2    -4.98            0             1
#> 3    -2.90            0.1            1
#> 4    -0.918           0.2            1
#> 5    -0.462           0.2            0.9
#> 6    -0.395           0.2            0.8
#> # ... with 16 more rows

eval_data %>%
  yardstick::accuracy(truth = factor(outcome, levels=levs),
    estimate = factor(pred_hard, levels=levs))
#> # A tibble: 1 x 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>         <dbl>

```

```

#> 1 accuracy binary          0.55

# Notes:
# - the yardstick packages wants outcome variables and hard predictions be
#   "factors" with the outcome of interest as the first level

#-----#
# Radial basis SVM tuning parameter selection
#-----#
#:: create function to fit, predict, and evaluate
eval_svm <- function(data_train, data_test, cost, gamma){

  #:: fit
  fit = svm(factor(class) ~ X1 + X2,
             data = data_train,
             #:: tuning parameters
             kernel = "radial",
             gamma = gamma,
             cost = cost
             )

  #:: predict
  pred = predict(fit, data_test, decision.values = TRUE)

  #:: evaluate
  eval_data = tibble(
    outcome = data_test$class,
    pred_hard = c(pred),
    pred_soft = as.numeric(attr(pred, "decision.values"))
  )

  levs = c("+1", "-1") # set outcome of interest at first level

  auROC = eval_data %>%
    yardstick::roc_auc(truth = factor(outcome, levels=levs), estimate = pred_soft)

  accuracy = eval_data %>%
    yardstick::accuracy(truth = factor(outcome, levels=levs),
                        estimate = factor(pred_hard, levels=levs))

  bind_rows(auROC, accuracy) %>%
    mutate(cost, gamma) # add tuning parameters
}

#:: test it out
eval_svm(
  data_train = data_mix[fold != 1,],
  data_test = data_mix[fold == 1,],
  cost = 1,
  gamma = .1
)
#> # A tibble: 2 x 5
#>   .metric .estimator .estimate cost gamma
#>   <chr>    <chr>         <dbl> <dbl> <dbl>
#> 1 roc_auc binary          1         1    0.1
#> 2 accuracy binary      0.95         1    0.1

```



```

# using tidymodels
library(rsample)
eval_svm(
  data_train = training(cv_folds$splits[[1]]),
  data_test = testing(cv_folds$splits[[1]]),
  cost = 1,
  gamma = .1
)
#> # A tibble: 2 x 5
#>   .metric .estimator .estimate cost gamma
#>   <chr>    <chr>         <dbl> <dbl> <dbl>
#> 1 roc_auc binary          1      1    0.1
#> 2 accuracy binary        0.95    1    0.1

# create grid of tuning parameter values
tune_grid = expand_grid(
  cost = 10^(-2:2),
  gamma = c(.1, .5, 1)
)

# loop over folds and grids
out = tibble()

for(k in unique(fold)){
  for(i in 1:nrow(tune_grid)){
    metrics = eval_svm(
      data_train = data_mix[fold != k,],
      data_test = data_mix[fold == k,],
      cost = tune_grid$cost[i],
      gamma = tune_grid$gamma[i]
    )

    out = bind_rows(out, metrics %>% mutate(fold = k) )
  }
}

# average over folds
out %>%
  # calculate average performance over folds
  group_by(.metric, cost, gamma) %>%
  summarize(mu = mean(.estimate), .groups = "drop") %>%
  # spread data wider to see both metrics
  pivot_wider(names_from = .metric, values_from = mu) %>%
  # arrange by auc and then accuracy (if ties)
  arrange(desc(roc_auc), desc(accuracy))
#> # A tibble: 15 x 4
#>   cost gamma accuracy roc_auc
#>   <dbl> <dbl>    <dbl>    <dbl>
#> 1  10    0.1    0.955    0.995
#> 2  0.1    0.1    0.95    0.995
#> 3  1     0.1    0.95    0.995
#> 4  0.01   0.1    0.62    0.995
#> 5 100    0.1    0.96    0.995
#> 6  0.1    0.5    0.955    0.992

```

```
#> # ... with 9 more rows

# Notes:
# - can go back and modify grid to focus on best tuning region

# using tidymodels and purrr

# function to search over grid
eval_svm_grid <- function(data, grid){
  pmap_df(grid, function(cost, gamma) eval_svm(training(data), testing(data), cost, gamma))
}

# try it out
data = cv_folds$splits[[1]]
eval_svm_grid(data, head(tune_grid))
#> # A tibble: 12 x 5
#>   .metric .estimator .estimate cost gamma
#>   <chr>    <chr>         <dbl> <dbl> <dbl>
#> 1 roc_auc  binary             1  0.01  0.1
#> 2 accuracy binary         0.5  0.01  0.1
#> 3 roc_auc  binary             1  0.01  0.5
#> 4 accuracy binary         0.5  0.01  0.5
#> 5 roc_auc  binary             1  0.01  1
#> 6 accuracy binary         0.5  0.01  1
#> # ... with 6 more rows

# loop over folds
map_df(cv_folds$splits, ~eval_svm_grid(data=.x, grid=tune_grid), .id="fold")
#> # A tibble: 300 x 6
#>   fold .metric .estimator .estimate cost gamma
#>   <chr> <chr>    <chr>         <dbl> <dbl> <dbl>
#> 1 1 roc_auc  binary             1  0.01  0.1
#> 2 1 accuracy binary         0.5  0.01  0.1
#> 3 1 roc_auc  binary             1  0.01  0.5
#> 4 1 accuracy binary         0.5  0.01  0.5
#> 5 1 roc_auc  binary             1  0.01  1
#> 6 1 accuracy binary         0.5  0.01  1
#> # ... with 294 more rows
```