Machine Learning

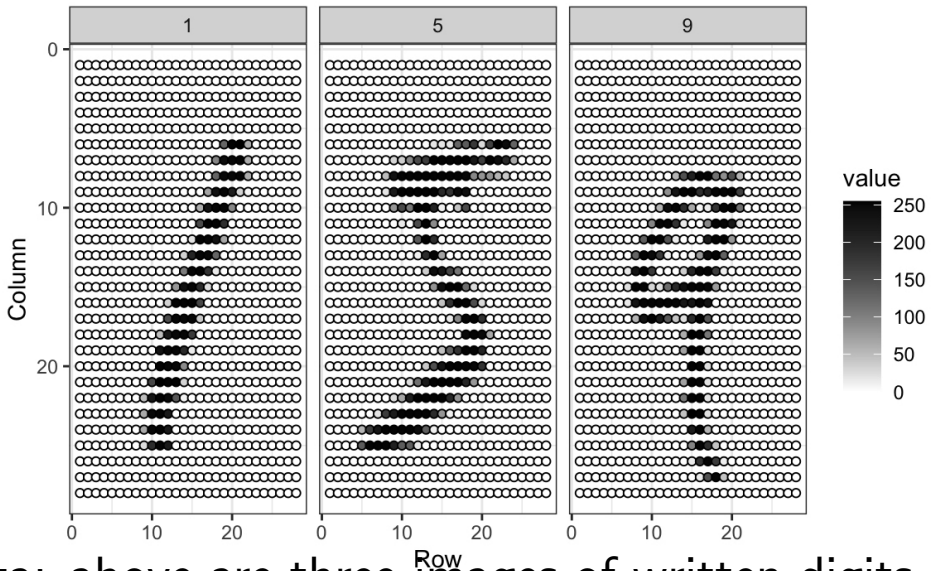
Department of Statistics, University of South Carolina

Stat 705: Data Analysis II

An Example: Zip Code Reader

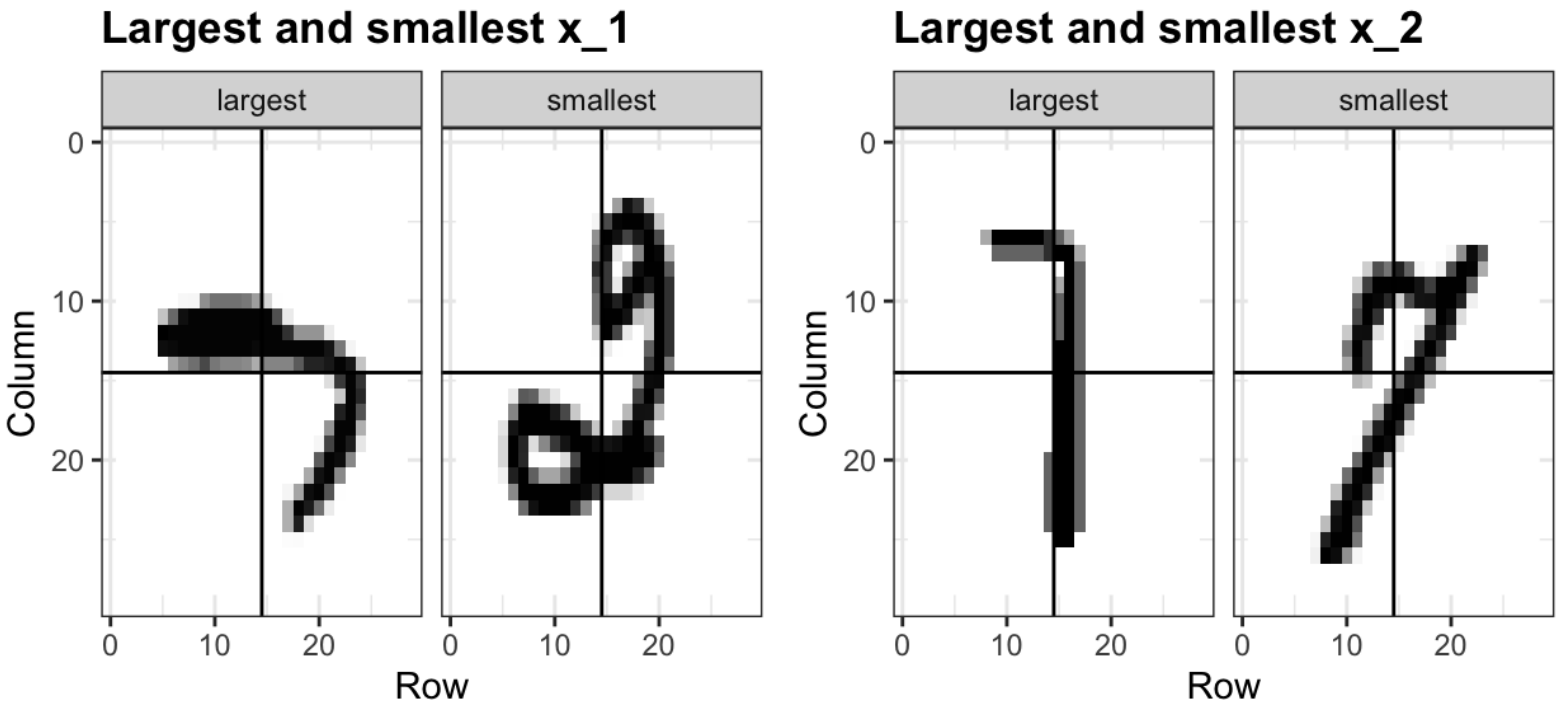
- Machine learning algorithms can help read zip codes and then a robot sorts the letters.
- In this section, we will learn how to build algorithms that can read a digit.

Pixels Data



- Training data: above are three images of written digits. These have been read by human and assigned an outcome Y.
- The images are converted into $28 \times 28 = 784$ pixels.
- For each pixel, we obtain a grey scale intensity between 0 (white) and 255 (black).
- For each digitized image i, we have a categorical outcome Y_i can be (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) and features (covariates) $X_{i1}, X_{i2}, ..., X_{i784}$. Let $\mathbf{X}_i = (X_{i1}, X_{i2}, ..., X_{i784})$

Case study: is it a 2 or a 7?



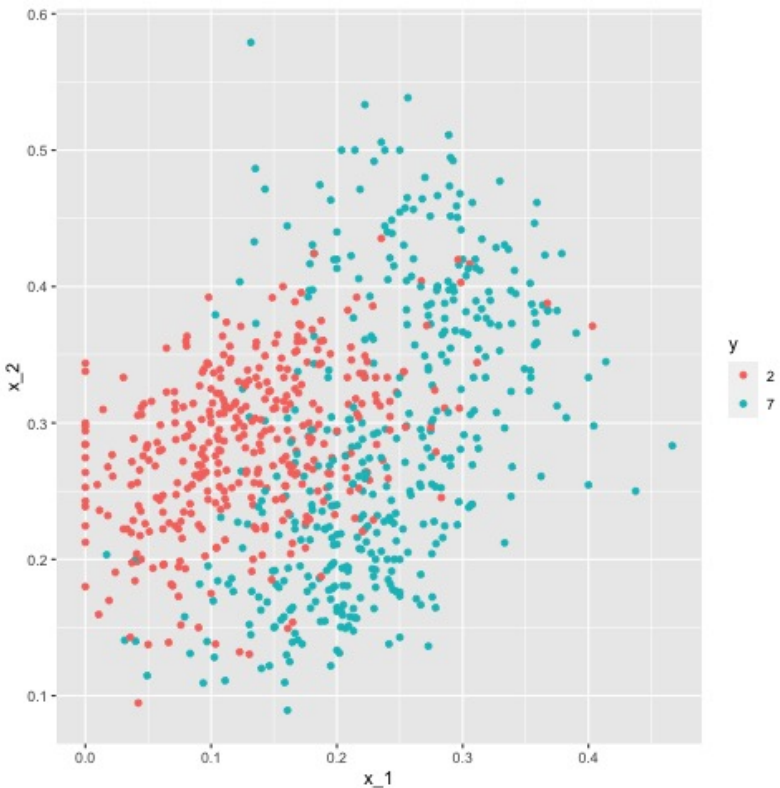
- X_1:the proportion of dark pixels that are in the upper left quadrant
- X_2: the proportion of dark pixels that are in the lower right quadrant

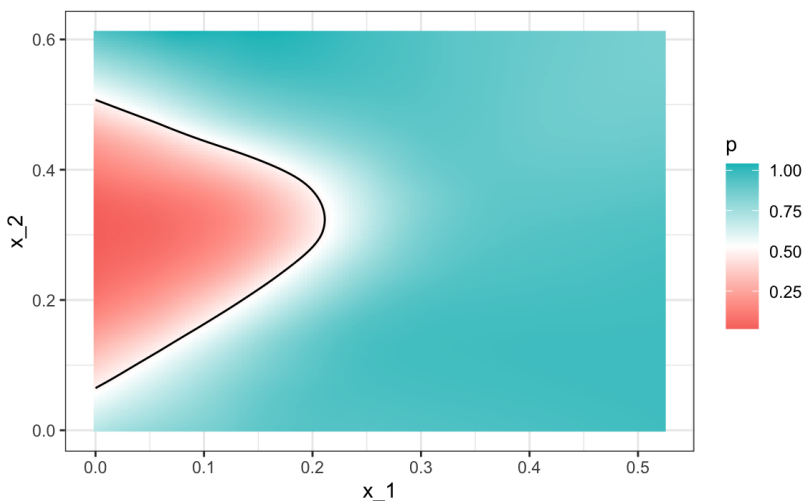
Case study: is it a 2 or a 7?

```
> library(tidyverse)
> library(dslabs)
> data("mnist_27")
> str(mnist_27)
List of 5
 $ train      :'data.frame': 800 obs. of  3 variables:
  ..$ y  : Factor w/ 2 levels "2","7": 1 2 1 1 2 1 2 2 2 1 ...
  ..$ x_1: num [1:800] 0.0395 0.1607 0.0213 0.1358 0.3902 ...
  ..$ x_2: num [1:800] 0.1842 0.0893 0.2766 0.2222 0.3659 ...
 $ test       :'data.frame': 200 obs. of  3 variables:
  ..$ y  : Factor w/ 2 levels "2","7": 1 2 2 2 2 1 1 1 1 2 ...
  ..$ x_1: num [1:200] 0.148 0.283 0.29 0.195 0.218 ...
  ..$ x_2: num [1:200] 0.261 0.348 0.435 0.115 0.397 ...
```

Zip Code Data

```
> train<-mnist_27$train
> plot(train$x_1, train$x_2, col=train$y)
```

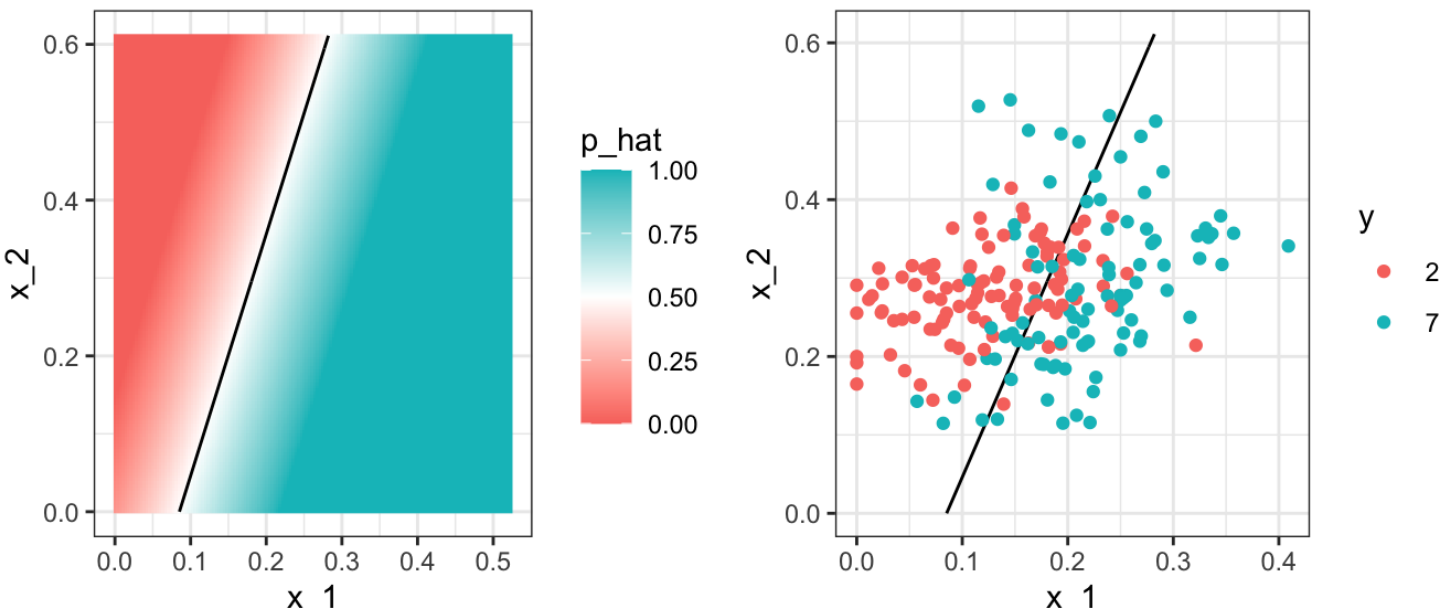




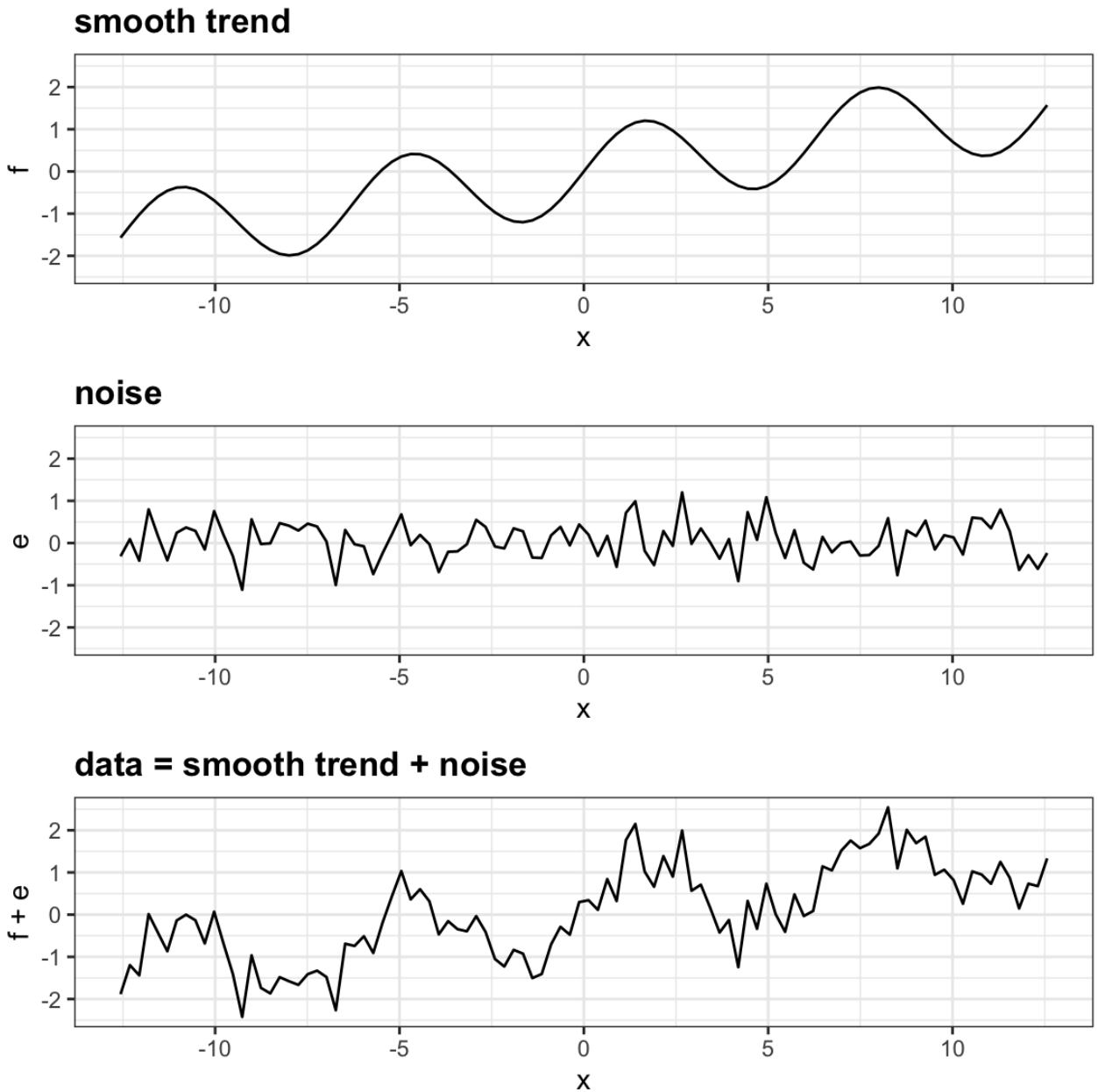
Logistic Regression

$$\log[\frac{P(Y=1)}{1-P(Y=1)}] = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

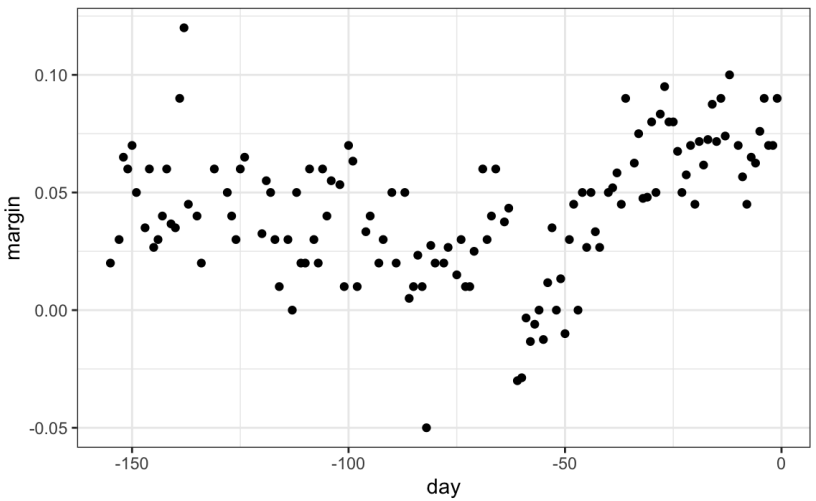
```
> train<-mnist_27$train
> train$y<-ifelse(as.numeric(train$y)==2, 1, 0)
> fit_glm<-glm(y ~ x_1 + x_2, data=train, family="binomial")
> p_hat_lm <- predict(fit_glm, mnist_27$test)
> y_hat_lm <- factor(ifelse(p_hat_lm > 0.5, 7, 2))
> confusionMatrix(y_hat_lm, mnist_27$test$y)$overall["Accuracy"]
Accuracy
    0.76
>
> p_hat <- predict(fit_glm, newdata = mnist_27$true_p, type = "response")
```

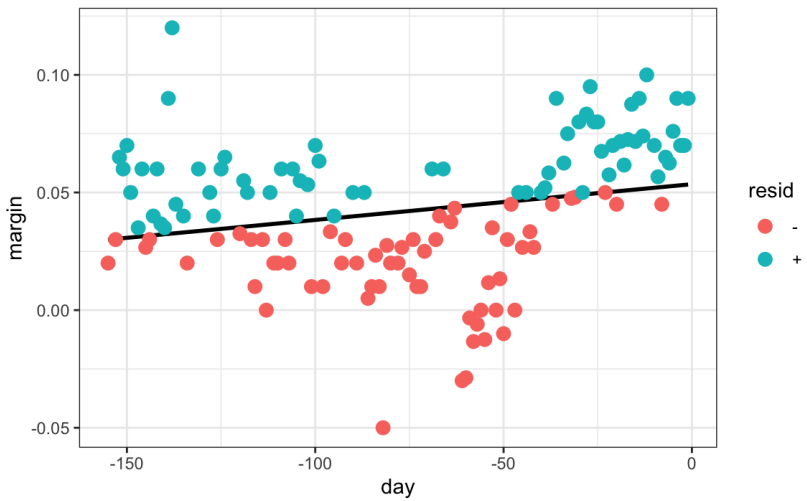


Smoothing

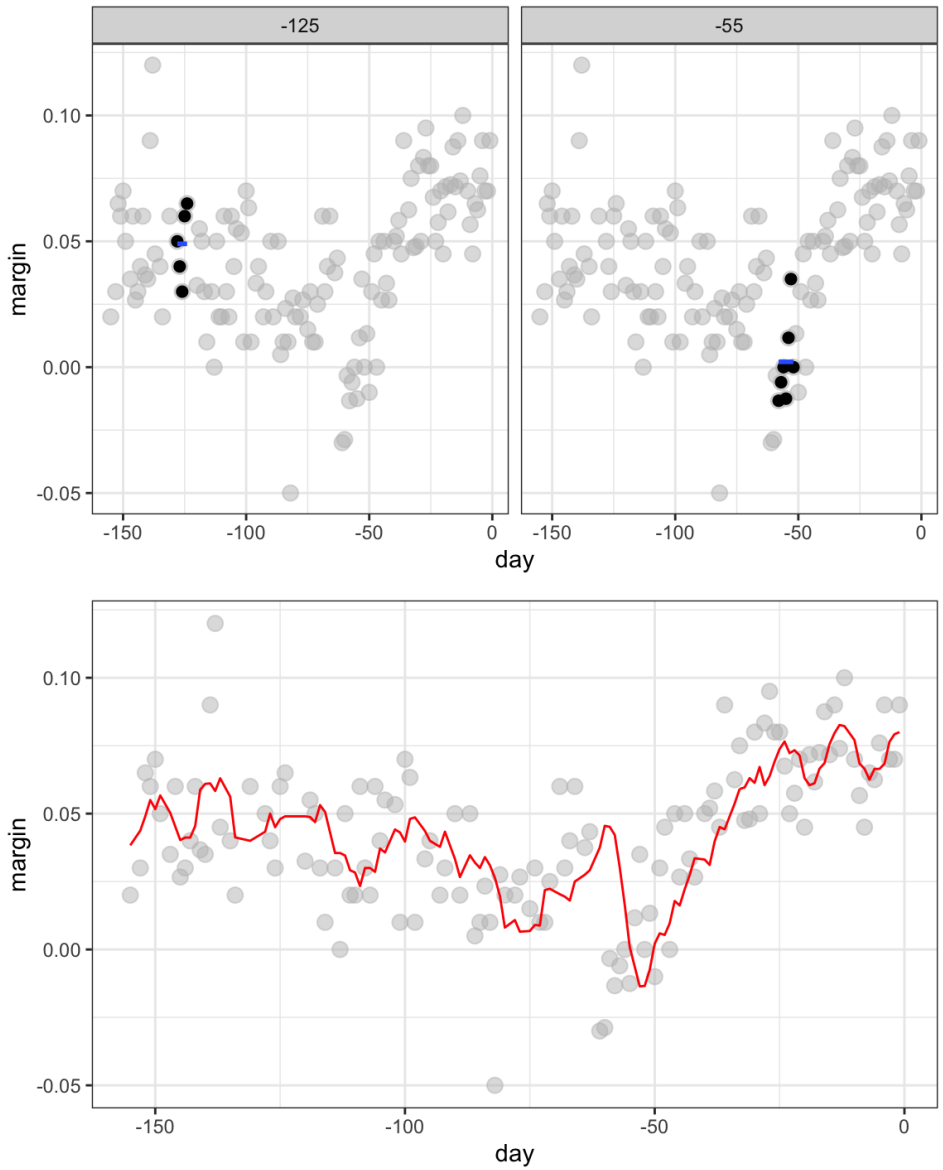


Smoothing: 2008 US Vote Poll Margin



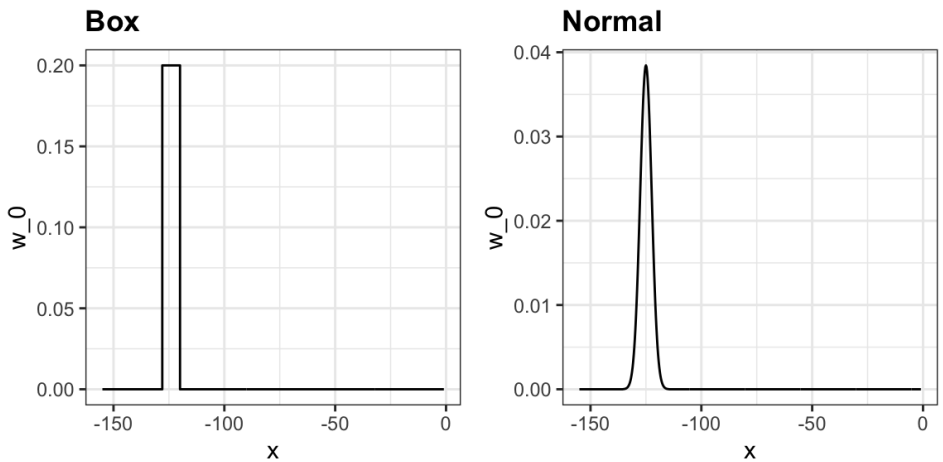


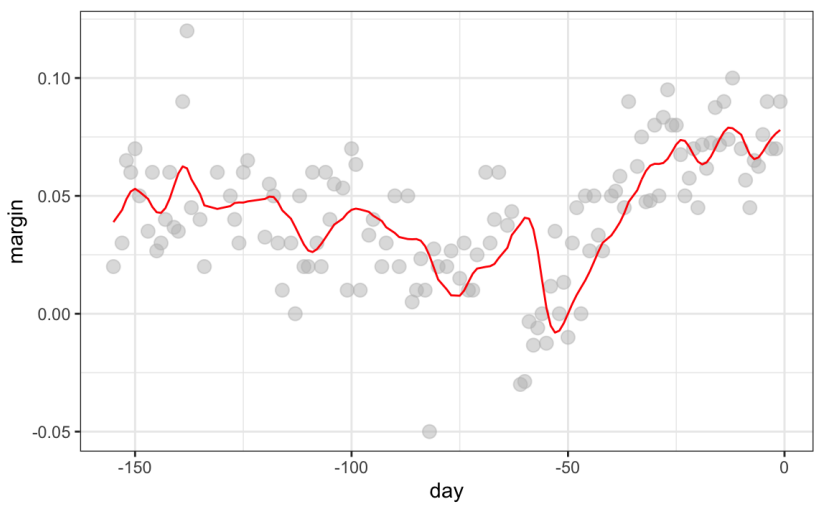
Bin smoothing



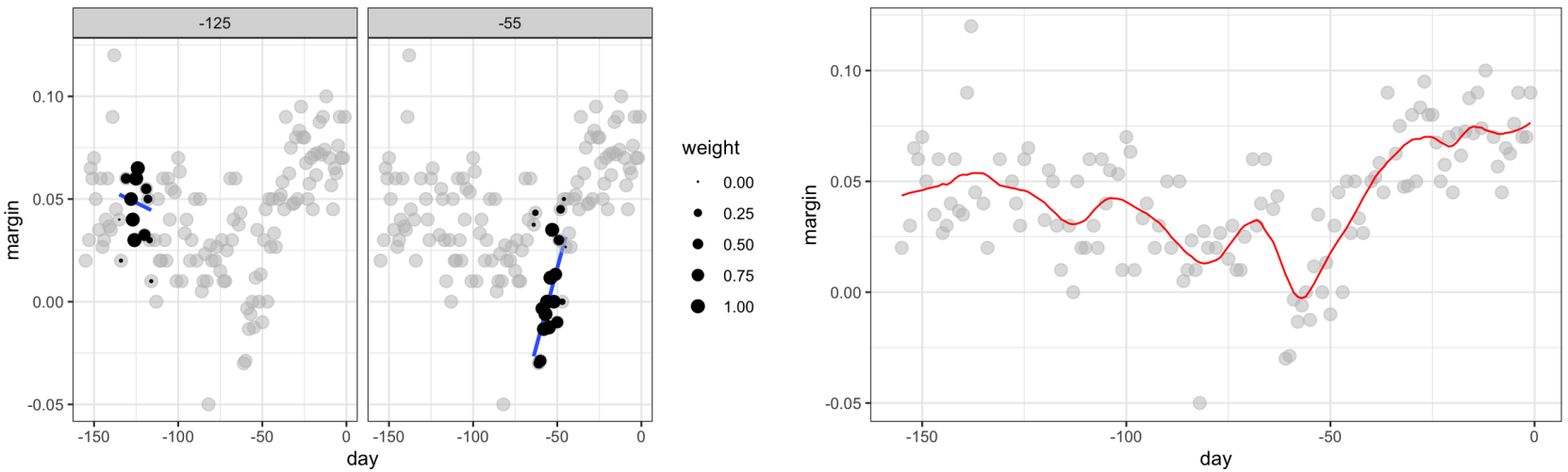
Kernel smoothing

$$\widehat{f(x_0)} = \sum_{i=1}^{n} w_0(x_i) Y_i$$



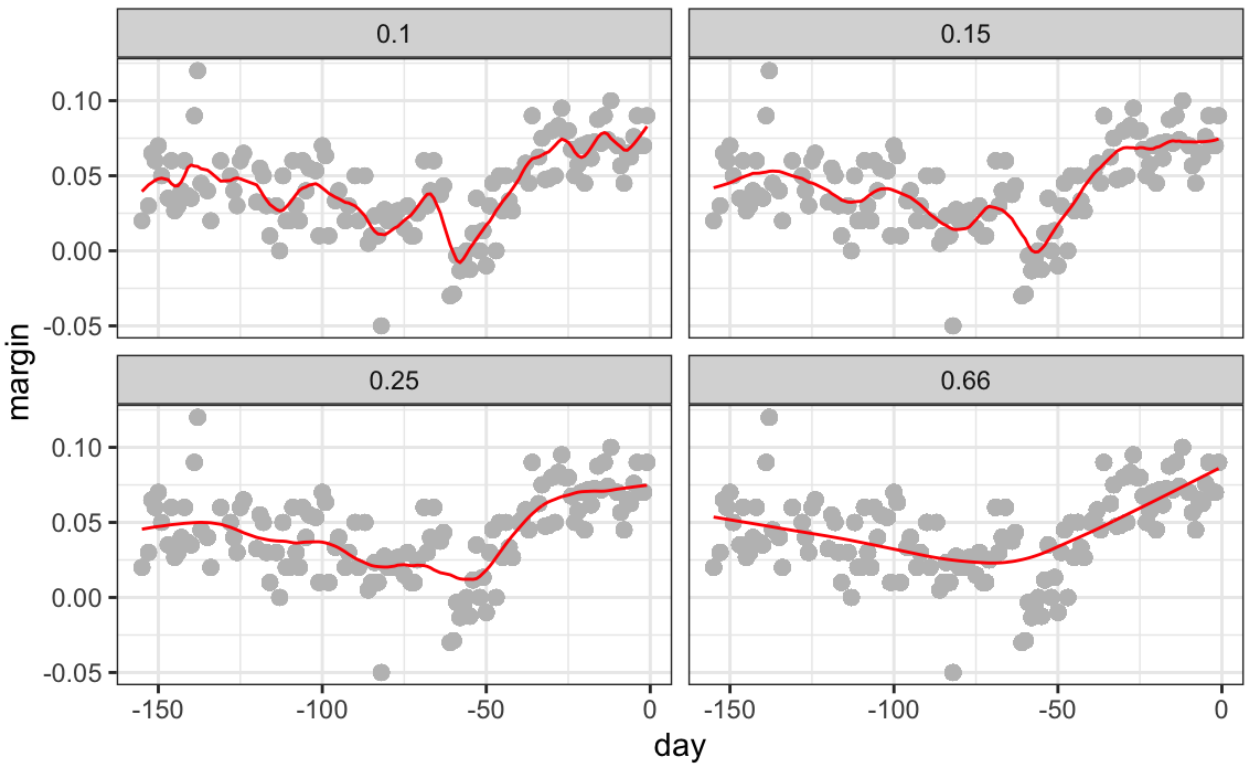


Local weighted regression (loess)



Loess

```
>total_days <- diff(range(polls_2008$day))
>span <- 21/total_days
>fit <- loess(margin ~ day, degree=1, span = span, data=polls_2008)
```



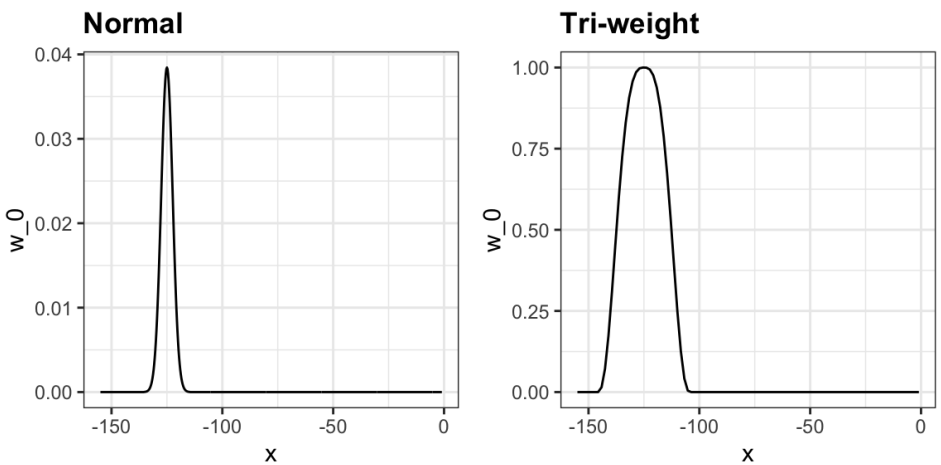
Loess estimates

$$\sum_{i=1}^{N} w_0(x_i)[Y_i - \beta_0 + \beta_1(x_i - x_0)]^2$$

Tukey tri-weight

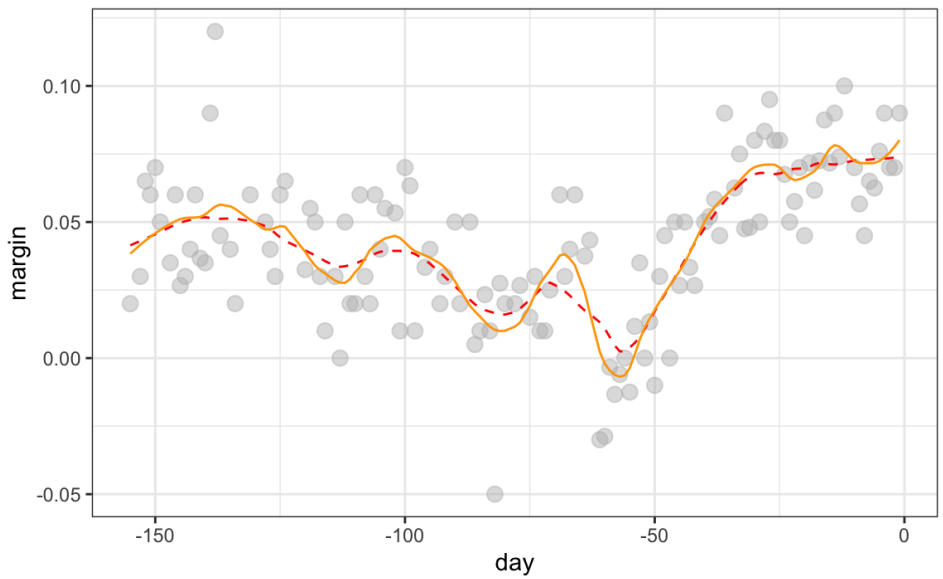
$$W(u) = (1 - |u|^3)^3 \text{ if } |u < -1| \text{ and } W(u) = 0 \text{ if } |u > 1|$$

$$w_0(x_i) = W(\frac{x_i - x_0}{h})$$

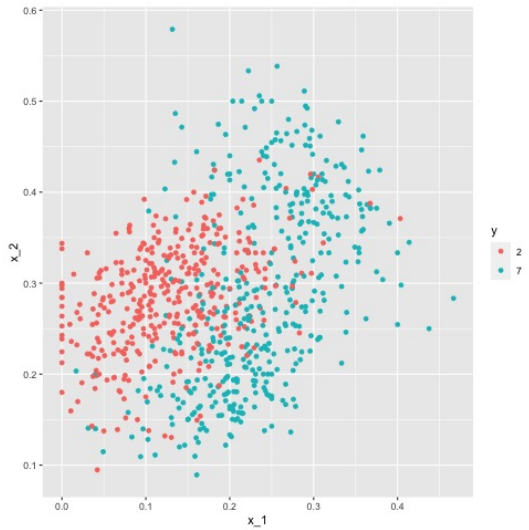


Loess

```
>total_days <- diff(range(polls_2008$day))
>span <- 21/total_days
>fit <- loess(margin ~ day, degree=1, span = span, data=polls_2008)
```

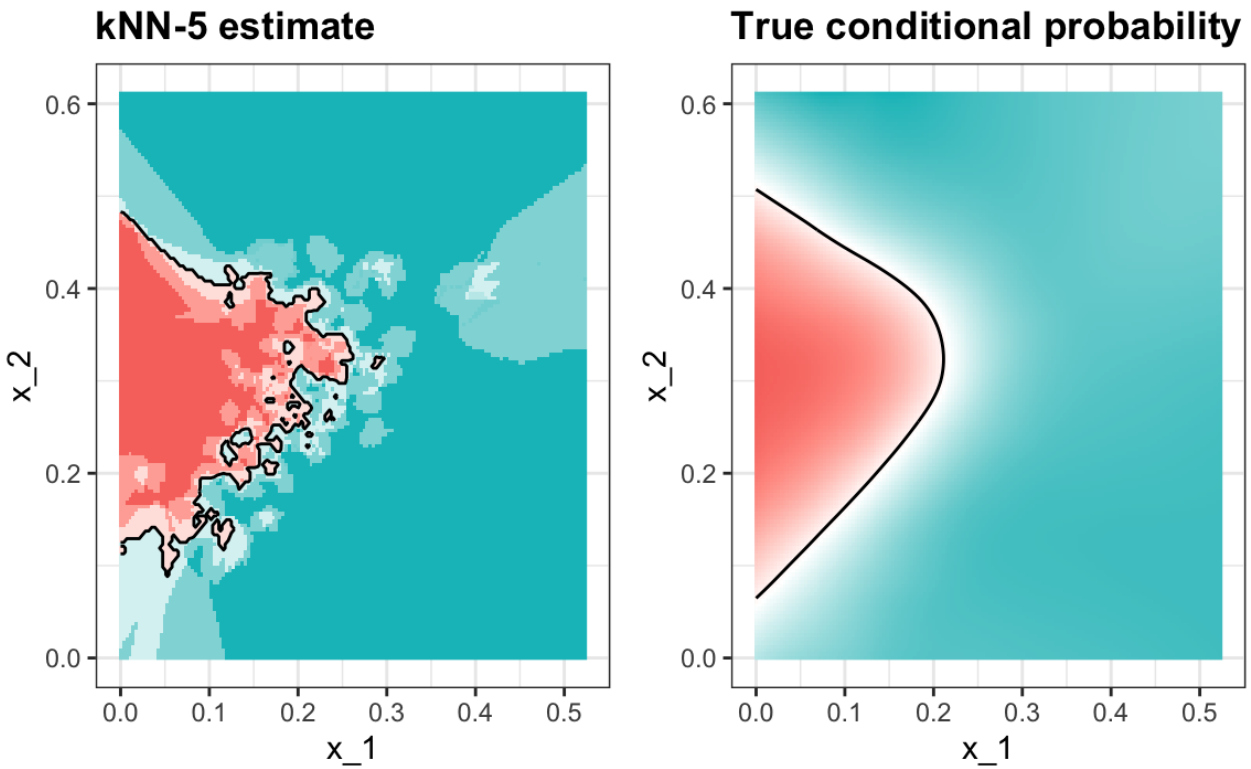


K-Nearest Neighbors (KNN)



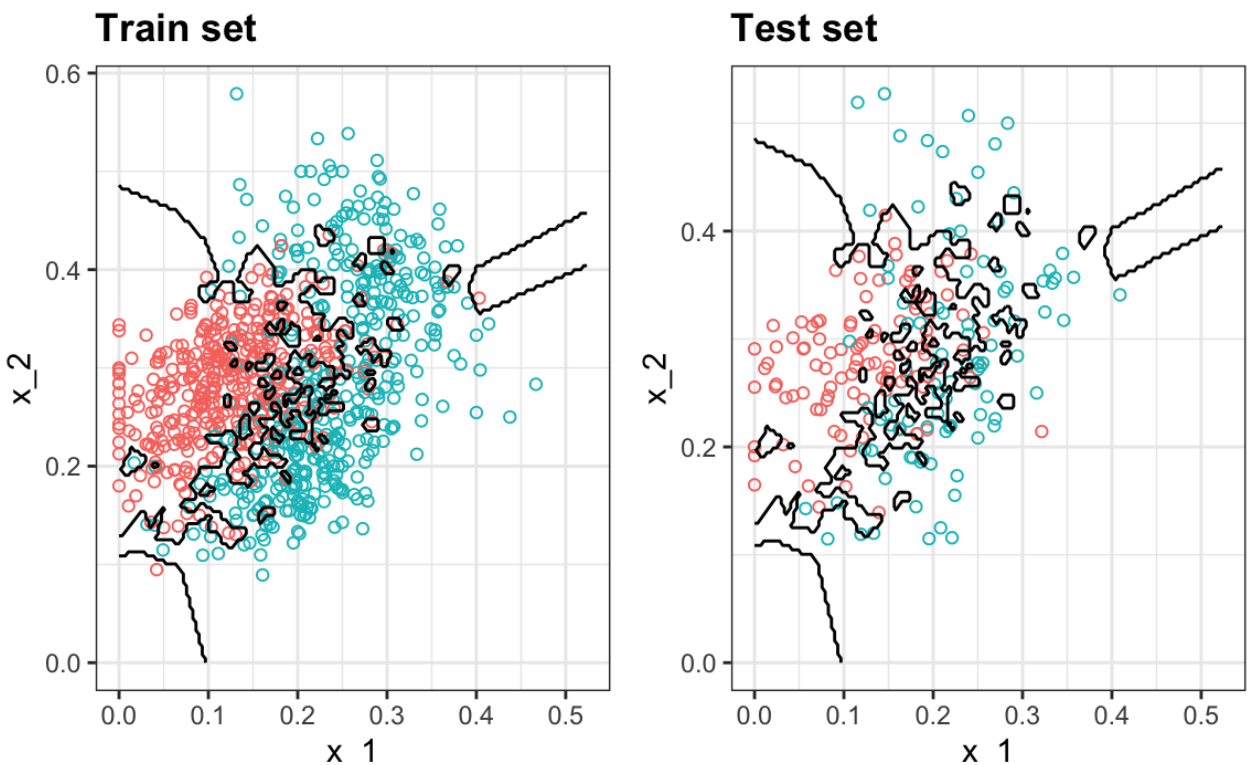

```
> library(caret)
> knn_fit <- knn3(y ~ ., data = mnist_27$train, k = 5)
> y_hat_knn <- predict(knn_fit, mnist_27$test, type = "class")
> confusionMatrix(y_hat_knn, mnist_27$test$y)$overall["Accuracy"]
Accuracy
   0.815
> fit_glm<-glm(y ~ x_1 + x_2, data=train, family="binomial")
> p_hat_lm <- predict(fit_glm, mnist_27$test)
> y_hat_lm <- factor(ifelse(p_hat_lm > 0.5, 7, 2))
> confusionMatrix(y_hat_lm, mnist_27$test$y)$overall["Accuracy"]
Accuracy
    0.76
```

KNN


kNN-5 estimate
True conditional probability
x_2
x_1
0.0
0.2
0.4
0.6
0.1
0.3
0.5

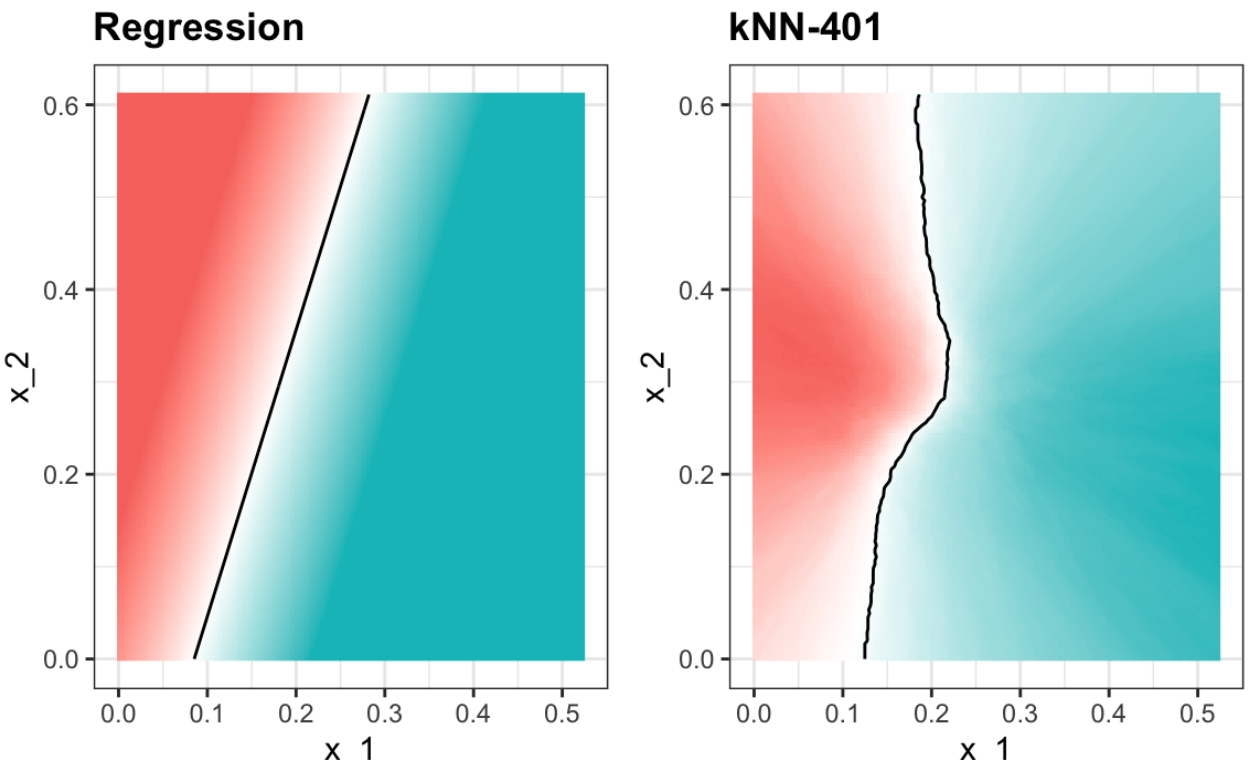
Overfitting

```
> y_hat_knn <- predict(knn_fit, mnist_27$train, type = "class")
> confusionMatrix(y_hat_knn, mnist_27$train$y)$overall["Accuracy"]
Accuracy
  0.8825
>
> y_hat_knn <- predict(knn_fit, mnist_27$test, type = "class")
> confusionMatrix(y_hat_knn, mnist_27$test$y)$overall["Accuracy"]
Accuracy
   0.815
```

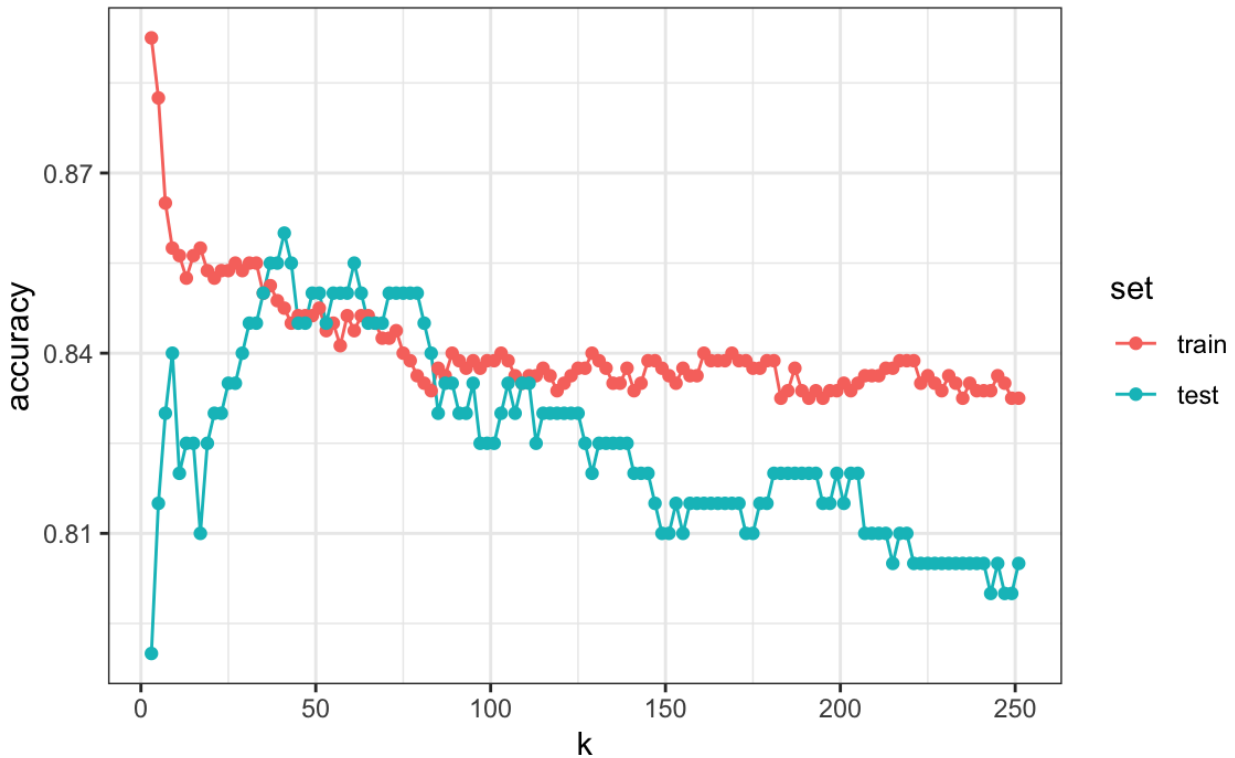


Over-smoothing

```
> knn_fit_401 <- knn3(y ~ ., data = mnist_27$train, k = 401)
> y_hat_knn_401 <- predict(knn_fit_401, mnist_27$test, type = "class")
> confusionMatrix(y_hat_knn_401, mnist_27$test$y)$overall["Accuracy"]
Accuracy
    0.79
```



Picking k in KNN

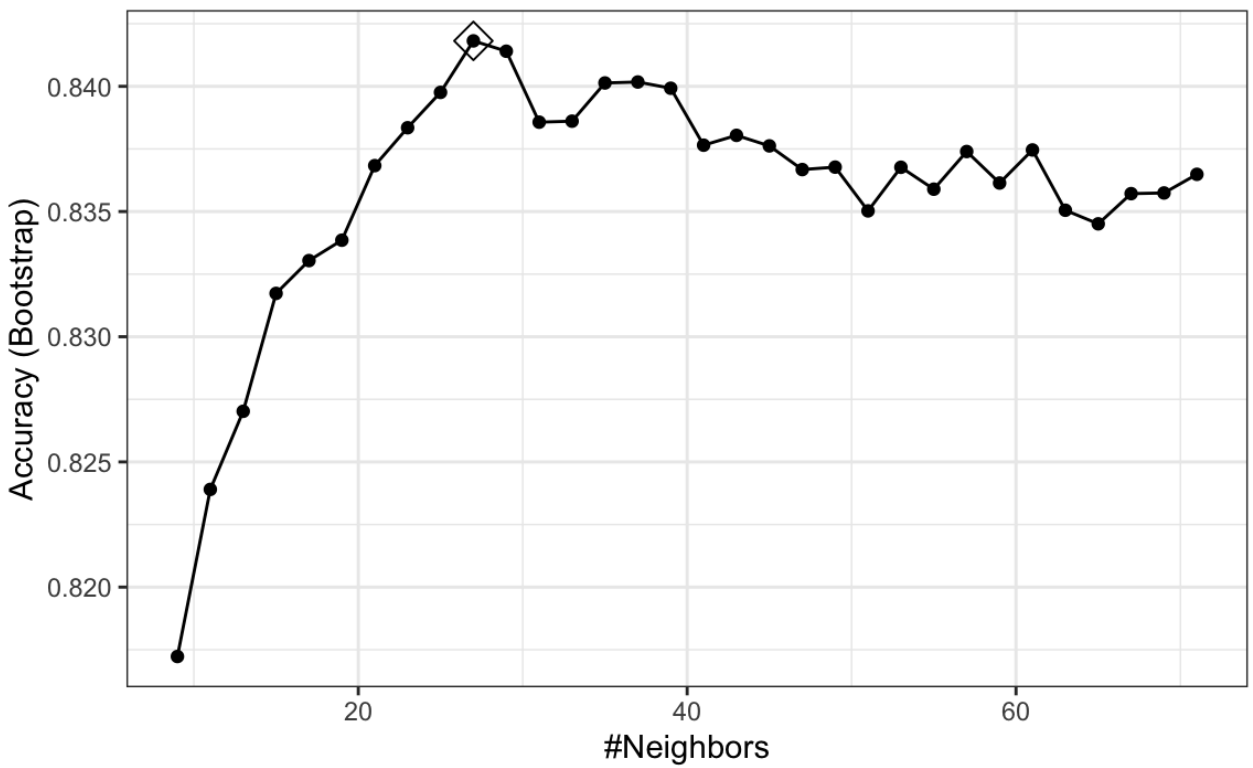


The caret package: train

```
> library(caret)
> train_glm <- train(y ~ ., method = "glm", data = mnist_27$train)
> train_knn <- train(y ~ ., method = "knn", data = mnist_27$train)
> y_hat_glm <- predict(train_glm, mnist_27$test, type = "raw")
> y_hat_knn <- predict(train_knn, mnist_27$test, type = "raw")
>
> confusionMatrix(y_hat_glm, mnist_27$test$y)$overall[["Accuracy"]]
[1] 0.75
>
> confusionMatrix(y_hat_knn, mnist_27$test$y)$overall[["Accuracy"]]
[1] 0.84
```

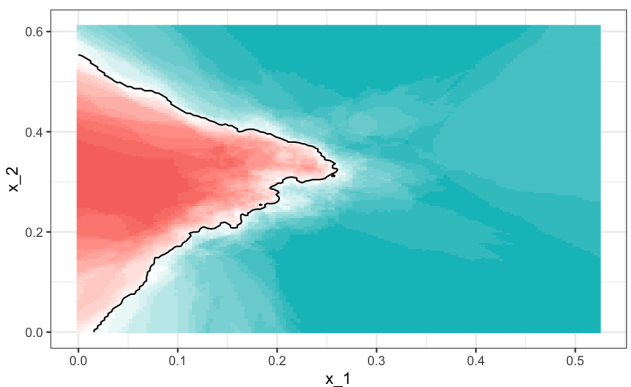
train KNN

```
> set.seed(2022)
> train_knn <- train(y ~ ., method = "knn",
+         data = mnist_27$train,
+         tuneGrid = data.frame(k = seq(9, 71, 2)))
> ggplot(train_knn, highlight = TRUE)
```



Best K

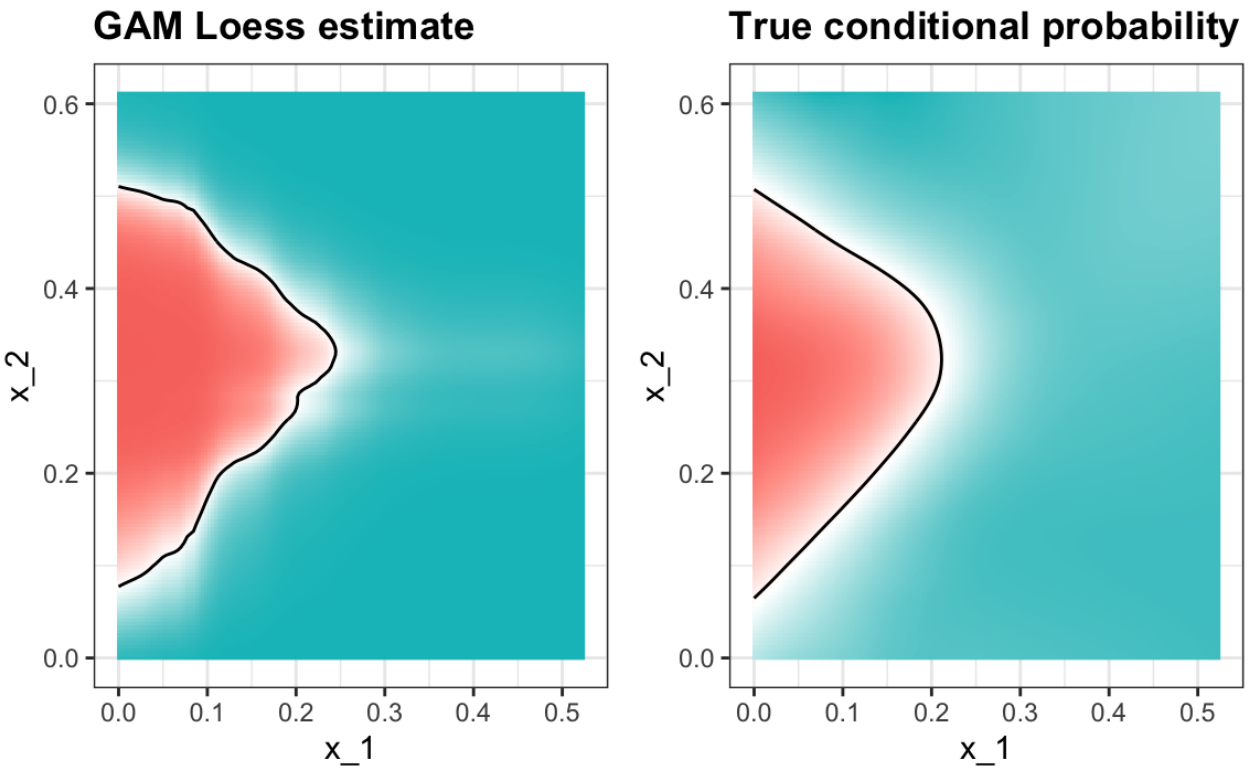
```
> train_knn$bestTune
    k
10 27
> train_knn$finalModel
27-nearest neighbor model
Training set outcome distribution:

  2   7
379 421
> confusionMatrix(predict(train_knn, mnist_27$test, type = "raw"),
+                 mnist_27$test$y)$overall["Accuracy"]
Accuracy
   0.835
```



Using Kernels

```
> modelLookup("gamLoess")
     model parameter  label forReg forClass probModel
1 gamLoess      span   Span   TRUE     TRUE      TRUE
2 gamLoess    degree Degree   TRUE     TRUE      TRUE
>  grid <- expand.grid(span = seq(0.15, 0.65, len = 10), degree = 1)
>  train_loess <- train(y ~ .,
+                      method = "gamLoess",
+                      tuneGrid=grid,
+                      data = mnist_27$train)
> confusionMatrix(data = predict(train_loess, mnist_27$test),
reference = mnist_27$test$y)$overall["Accuracy"]
Accuracy
    0.84
```

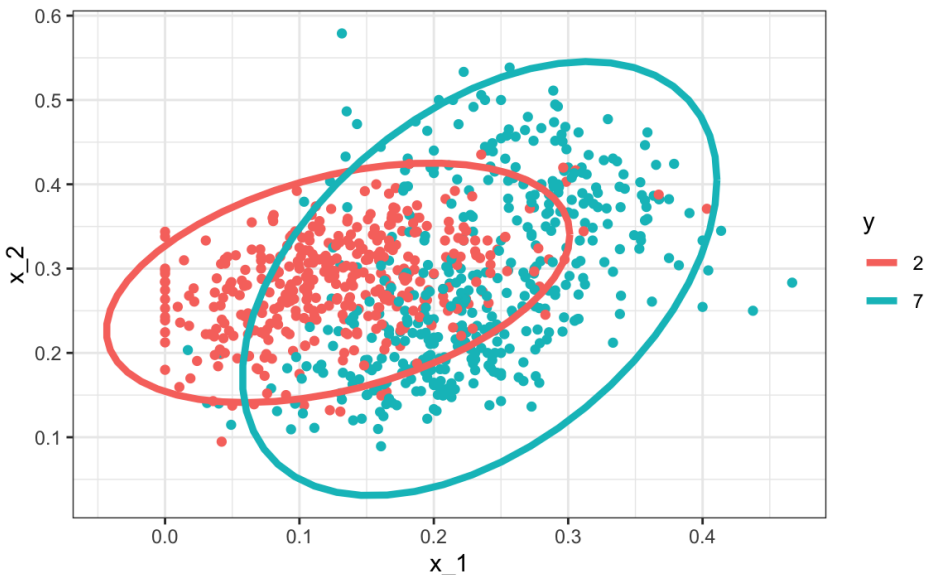
GAM Loess estimate



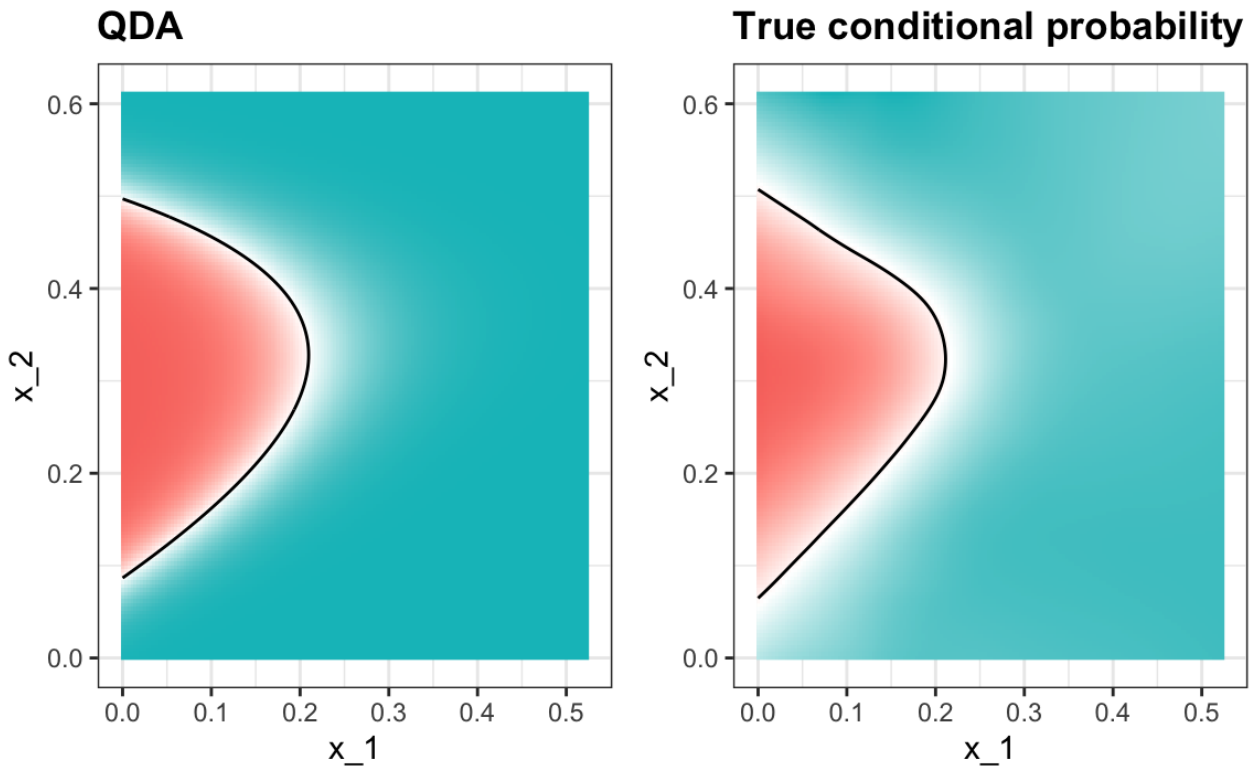
Quadratic Discriminant Analysis (QDA)

Bayes Rule

$$p(x) = Pr(Y = 1|X = x) = \frac{f_{X(x)|Y=1} P(Y = 1)}{f_{X|Y=0}(x)P(Y = 0) + f_{X|Y=1}(x)P(Y = 1)}$$

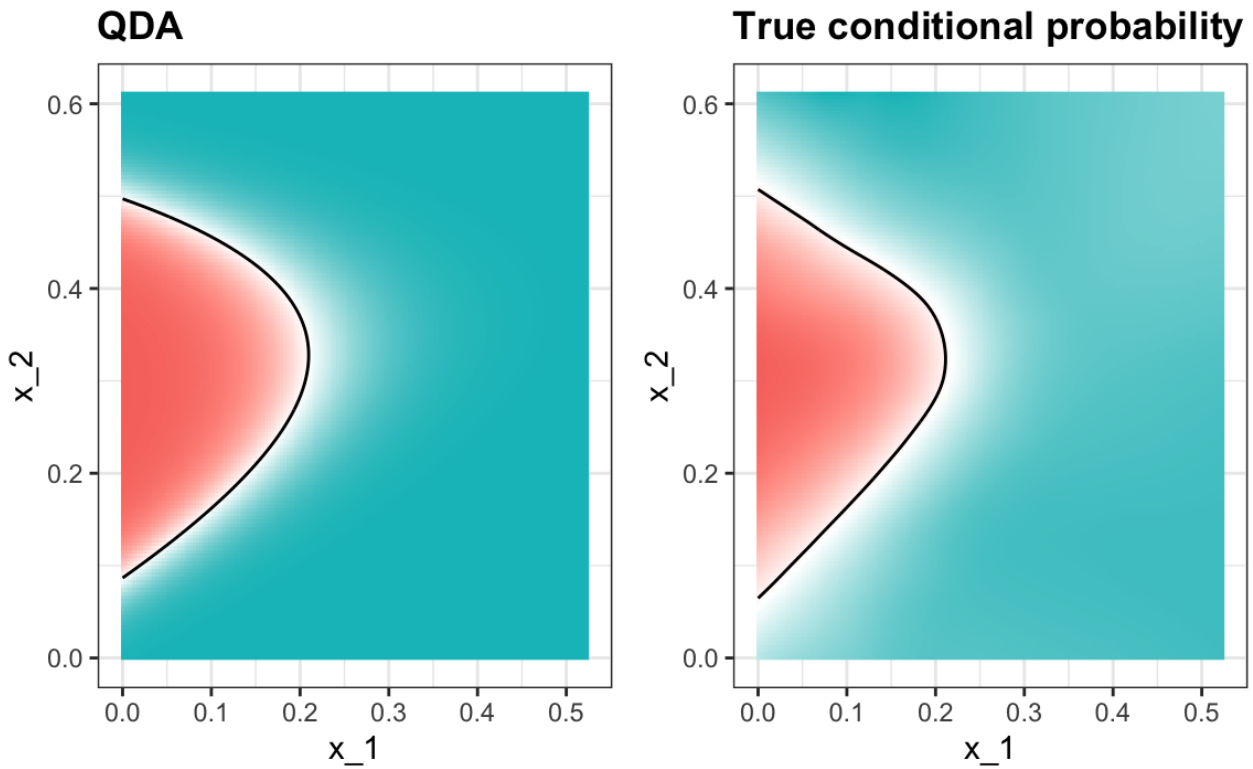


QDA



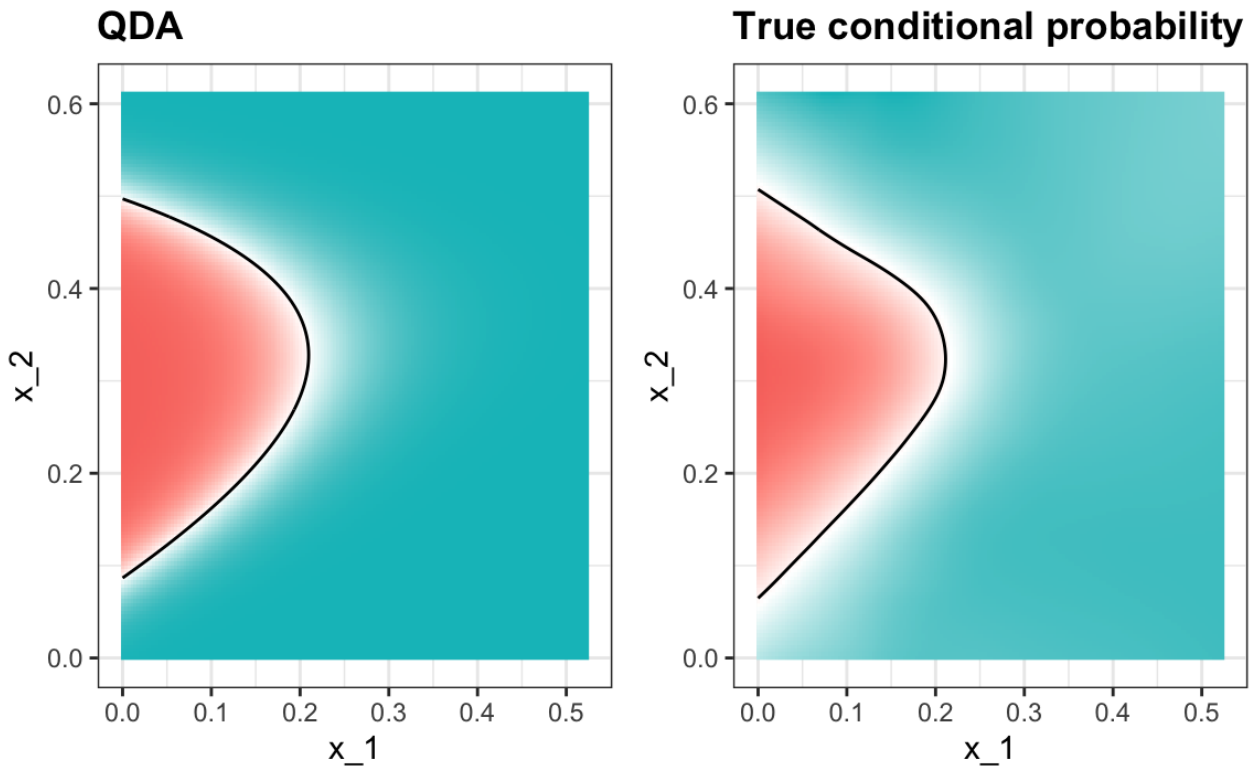
- QDA does not work as well as the kernel methods

QDA



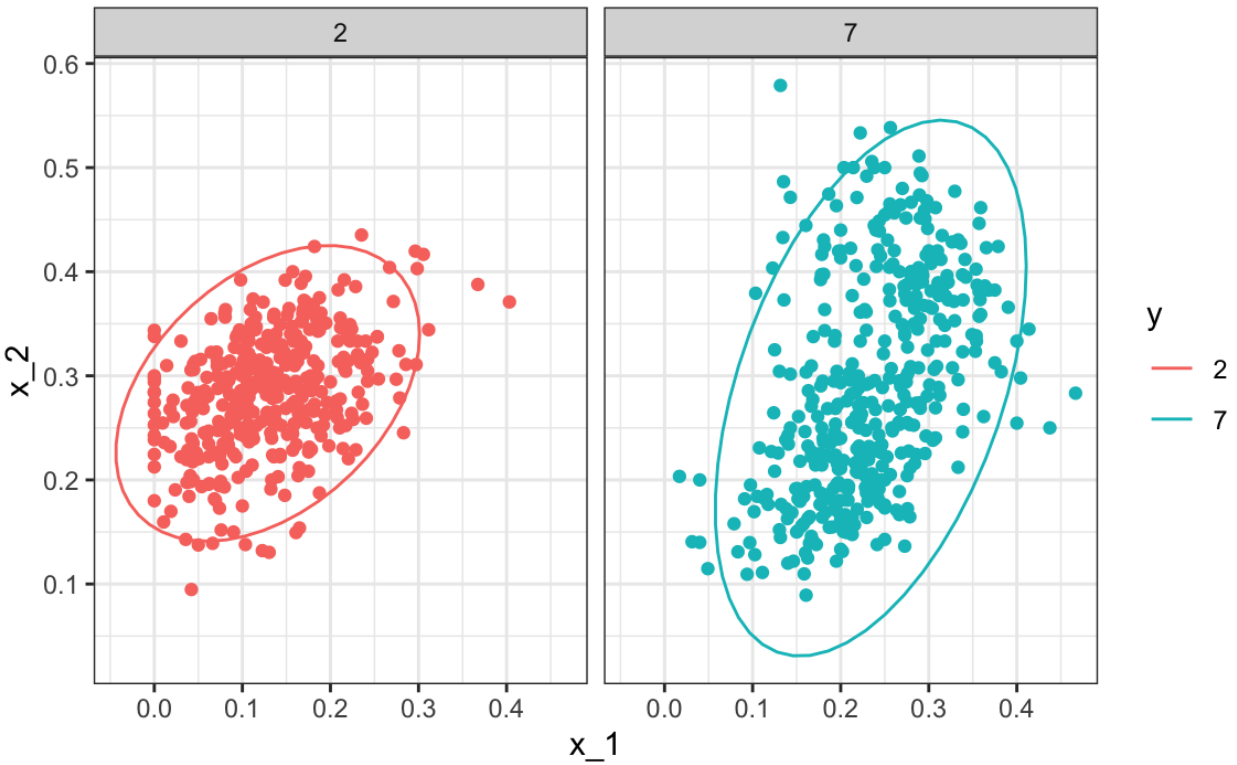
- QDA does not work as well as the kernel methods
- Normality assumption does not fit

QDA

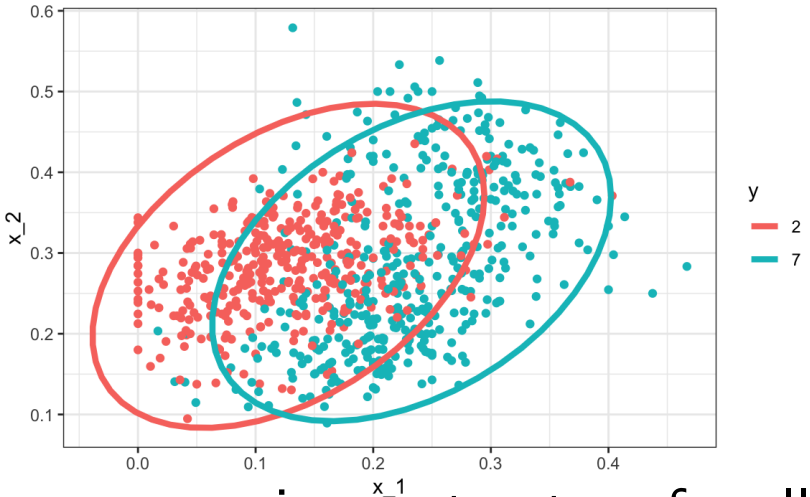


- QDA does not work as well as the kernel methods
- Normality assumption does not fit
- QDA becomes harder with the number of predictors increases. (How many parameters needed to be estimated if we have say, 10 predictors?)

Normality assumption



Linear Discriminant Analysis (LDA)



- Assume the same covariance structure for all classes to reduce the number of parameters needed to be estimated