# Classification: KNN and SVM

Data Science 101 Team

# Reading

Material from:

James, G., Witten, D., Hastie, T. and Tibshirani, R., 2021. An Introduction to Statistical Learning, Second Edition.

Referred to as ISLR in the following. Freely available online.

If you're interested: Chapter 4, 2.2, 9

# Recap: What is classification?

- The linear regression model assumes that the response $Y$ is quantitative.
- Often we have categorical data.
- We will look at approaches for predicting qualitative responses. Predicting a qualitative response is also referred to as "classification".
- We have seen classification before, in the context of logistic regression.

# Bayes Classifier

- Suppose our test error rate is $Ave(I(y_0 \neq \hat{y}_0))$ (i.e. the average number of times that our true test label is different from our prediction based on the test predictors $x_0$).
- A good classifier is the one for which the test error is smallest.
- Can show that the test error rate is minimized by the classifier that assigns each observation to the most likely class, given its predictor values.
- In math: Want to assign to the class $j$ for which $P(Y = j|X = x_0)$ is largest.
- This classifier is called the Bayes Classifier.

# Bayes Classifier

- We would like to predict qualitative responses using the Bayes Classifier (since it has the lowest possible error rate).
- But for real data we don't know the distribution of $Y$ given $X$, so we can't compute the Bayes classifier.
- What we can do is to *estimate*: We can classify an observation to the class with the highest *estimated* probability.

# KNN

- K-Nearest Neighbors (KNN) is a classification method.
- Suppose we have a test observation $x_0$.
- KNN identifies the $K$ points in the training data that are closest to $x_0$. Let's call the $N_0$.
- Suppose that our outcome $Y$ takes on classes $j = 1, ..J$.

# KNN

- KNN calculates

$$P(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_j = j)$$

.

- This looks complicated. What is this?
- It just calculates the fraction of times that the $K$ Nearest Neighbors to $x_0$ in the training set take on a particular category of the outcome.

# KNN Example

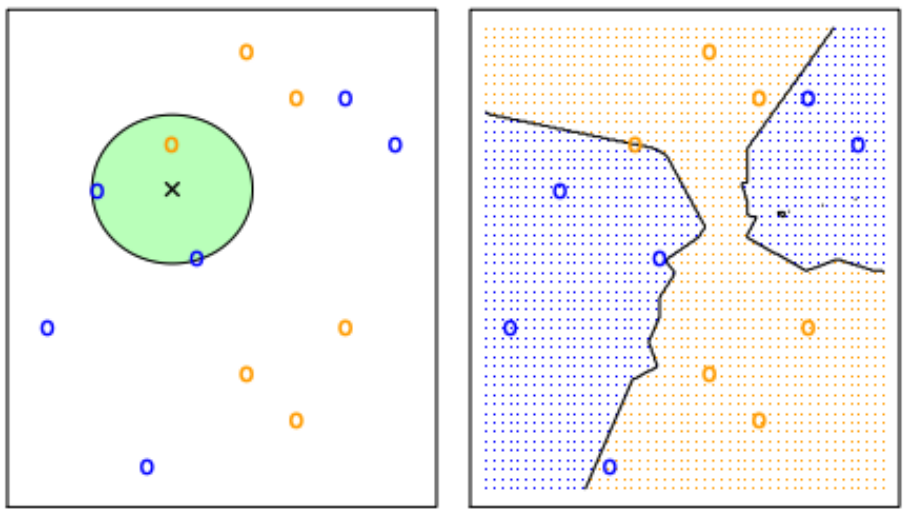


Figure 2.14 ISLR.

# KNN: Small K vs. Large K

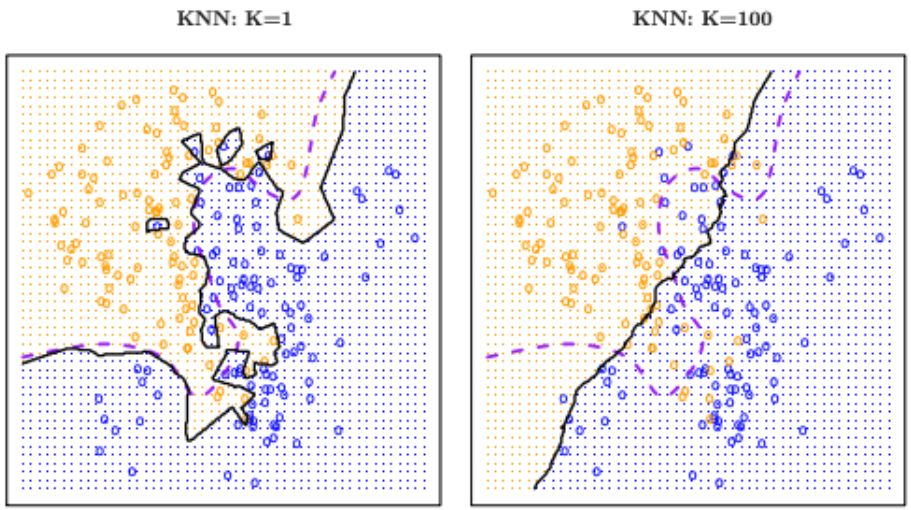



Figure 2.16 ISLR. Left panel: $K = 1$, right panel: $K = 100$.

With $K = 1$, the boundary is "overly flexible", while with $K = 100$, it is not sufficiently flexible. The "best possible" boundary (Bayes Classifier) is shown as purple dashed line.

# KNN Example

- Left picture: KNN with $K = 3$. The test observation is shown as a black cross. The three closest points are identified. We predict that the test observation belongs to the most commonly occurring class, in this case blue.
- Right picture: The "decision boundary". The blue grid shows the region in which a test observation would be assigned to the blue class, and the orange grid the region in which a test observation would be assigned to the orange class.

# KNN: How to use in R?

- Use the function `knn()`, which is part of the `class()` library.
- Data on stock market: Daily percentage returns for S&P 500 stock index between 2001 and 2005.

```
library(ISLR)
head(Smarket)
```

```
  Year   Lag1   Lag2   Lag3   Lag4   Lag5 Volume  Today Direction
1 2001  0.381 -0.192 -2.624 -1.055  5.010 1.1913  0.959        Up
2 2001  0.959  0.381 -0.192 -2.624 -1.055 1.2965  1.032        Up
3 2001  1.032  0.959  0.381 -0.192 -2.624 1.4112 -0.623      Down
4 2001 -0.623  1.032  0.959  0.381 -0.192 1.2760  0.614        Up
5 2001  0.614 -0.623  1.032  0.959  0.381 1.2057  0.213        Up
6 2001  0.213  0.614 -0.623  1.032  0.959 1.3491  1.392        Up
```

# Using KNN in R

```
library(class)
train <- (Smarket$Year < 2005)
train.X <- cbind (Smarket$Lag1 , Smarket$Lag2)[train , ]
test.X <- cbind (Smarket$Lag1 , Smarket$Lag2)[!train , ]
train.Direction <- Smarket$Direction[train]
Direction.2005 <- Smarket$Direction[!train]

set.seed (1)
knn.pred <- knn(train.X, test.X, train.Direction , k = 1)

table(knn.pred , Direction.2005)
```

```
        Direction.2005
knn.pred Down Up
    Down   43 58
    Up     68 83
```

```
(83 + 43) / 252
```

```
[1] 0.5
```

Not very good: Only 50% correctly predicted.

# Using KNN in R: Another Example

- `Caravan` data: 85 predictors that measure demographic characteristics on 5,822 individuals.
- The response is `Purchase`, indicating whether or not an individual bought a caravan insurance policy.

# Scaling and KNN

- KNN predicts the class by identifying observations that are *nearest* to a given test observation. `knn()` uses Euclidean distance.
- The scale of the predictors matter!
- Variables with a large scale will have a larger effect on the *distance* between the observations (and therefore on the KNN classifier).

# Scaling and KNN

- For example, consider *salary* and *age*. As far as KNN is concerned, a difference in $1,000$ Dollars in salary is enormous compared to a difference in 50 years in age.
- Therefore, `salary` will drive the KNN classification results, and age will have almost no effect.
- How to handle? *Standardize* the data so that all variables have mean 0 and standard deviation 1. This can be done in R using the `scale()` function.

# Caravan Example

```
standardized.X <- scale(Caravan[, -86])
# Did it standardize things?
var(Caravan[, 1])
```

```
[1] 165.0378
```

```
var(standardized.X[, 1])
```

```
[1] 1
```

```
mean(Caravan[, 1])
```

```
[1] 24.25335
```

```
mean(standardized.X[, 1])
```

```
[1] -7.025576e-17
```

# Caravan Example

```
set.seed(1)
knn.pred <- knn(train.X, test.X, train.Y, k = 1)
mean(test.Y != knn.pred)
```

```
[1] 0.118
```

# Caravan Example

But is this any good?

```
mean(test.Y != "No")
```

```
[1] 0.059
```

Only 6% of customers purchased insurance. So we could get the test error rate down to 6% by just predicting "No" regardless of the value of the predictors!

# Caravan Example

What about predicting the customers that are going to buy insurance?

```
table (knn.pred , test.Y)
```

```
        test.Y
knn.pred  No Yes
     No  873  50
     Yes  68   9
```

```
 9 / (68 + 9)
```

```
[1] 0.1168831
```

It turns out that we do better than random guessing, when we look at correctly predicting the customers who are going to buy insurance.

This becomes better when we use $k = 3$ or $k = 5$ instead.

# Support Vector machines

- Support Vector Machines (SVM) were developed in the computer science community in the 1990s.
- They have been shown to perform well in a variety of settings, and are often considered on of the "best out of the box" classifiers.

# Hyperplane - 2 Dimensions

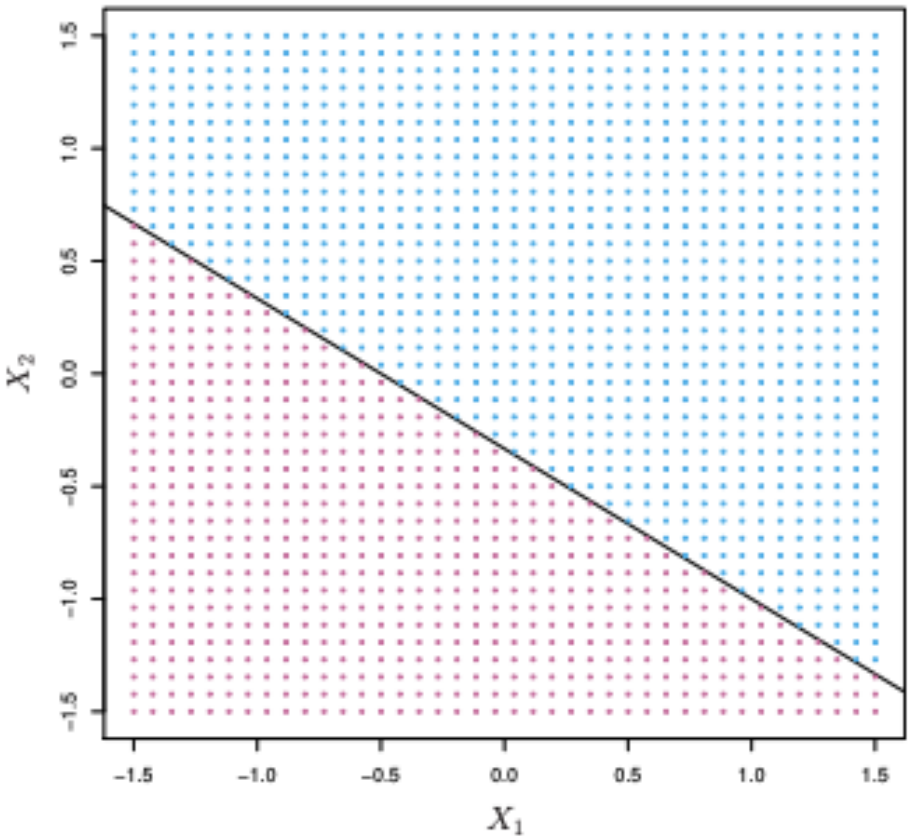



Figure 9.1 ISLR.

- The picture shows the "hyperplane" $1 + 2X_1 + 3X_2 = 0$. The blue region is the set of points for which $1 + 2X_1 + 3X_2 > 0$ and the purple region is the set of points for which $1 + 2X_1 + 3X_2 < 0$.
- We can extend the previous idea to multiple dimensions. You can think about it similar to a "linear decision boundary" in high-dimensional space (it has one dimension less than the entire space).

# Separating Hyperplanes

Suppose that it is possible to construct such a hyperplane that separates the training observations perfectly according to their class labels.

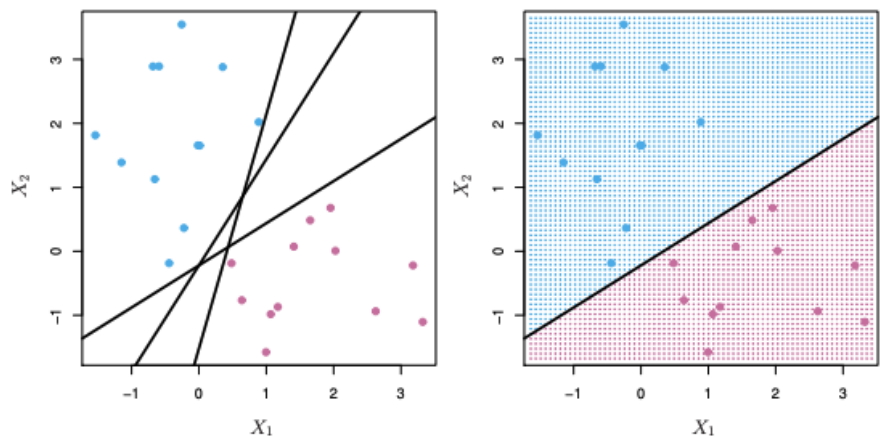



Figure 9.2 ISLR.

# Separating Hyperplanes - Previous Picture

- Two classes of observations: Blue and purple. Two predictors: $X_1$ and $X_2$.
- Left panel: Three separating hyperplanes (out of many possible) are shown in black.
- Right panel: Separating hyperplane shown in black. The blue and purple grid shows the decision rule made by a classifier based on this separating hyperplane.

# Separating Hyperplanes - Previous Picture

- If a separating hyperplane exists, we can use it to construct a very natural classifier: Assign to the class depending on which side of the hyperplane it is located.
- In the previous picture: A test observation falling into the blue portion of the grid will be assigned to the blue class, and a test observation falling into the purple portion of the grid will be assigned to the purple class.

# Separating Hyperplane - Classification

- Label the observations from the blue class as $y_i = 1$ and from the purple class as $y_i = -1$.
- Then the separating hyperplane has the property that: $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$ if $y_i = 1$ and $\beta_0 + \beta_1 X_1 + \beta_2 X_2 < 0$ if $y_i = -1$.
- So we can classify a test observation $x^*$ based on the sign: If $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^*$ is positive, assign to class 1, if negative, to class -1.

# Separating Hyperplane - Classification

- Can also use the magnitude: if $f(x^*)$ is far from zero, then this means that $x^*$ is far from the hyperplane, so we can be more confident about our class assignment for $x^*$.
- If $f(x^*)$ is close to zero, then $x^*$ is located near the hyperplane, so we are less certain about the class assignment for $x^*$.

# Maximum Margin Classifier

- We saw in the previous picture that if our data can be perfectly separated using a hyperplane, then there could be an infinite number of such hyperplanes!
- Why? We can usually shift a given separating hyperplane up or down a tiny bit (or rotate), without coming into contact with other observations.
- So we need some way to decide which hyperplane to use!

# Maximum Margin Hyperplane

- Idea: Look for the separating hyperplane that is farthest from the training observations. This is called the Maximum Margin Hyperplane!
- Compute the distance from each training observation to a given separating hyperplane.
- The *margin* is the smallest such distance (i.e. the minimal distance from the observations to the hyperplane).

# Maximum Margin Hyperplane

- The *Maximum Margin Hyperplane* then is the hyperplane for which the margin is largest (i.e. the hyperplane that has the farthest minimum distance to the training observations).
- Now we can classify a test observation based on which side of the maximal margin hyperplane it lies on.
- This is known as the *Maximal Margin Classifier*.
- We hope that the classifier that has large margin on the training data also has large margin on the test data. The maximal margin classifier is often succsessful, but can lead to overfitting when the number of predictors is large.

# Maximum Margin Hyperplane

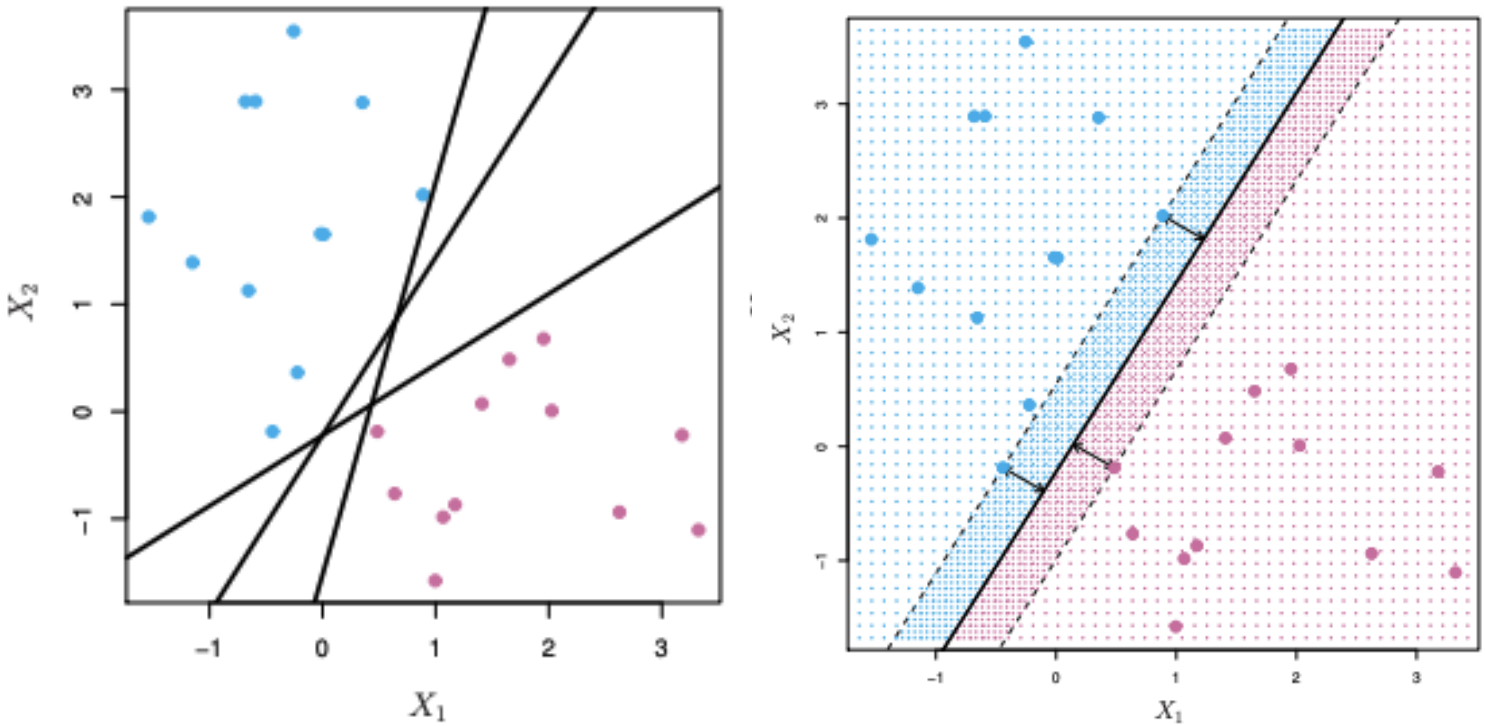



Figure 9.2 (i) and 9.3 in ISLR. On the right panel, the maximal margin hyperplane is shown as a solid line. The margin is the distance from the solid line to either of the dashed lines.

# Support Vectors

- We see that 3 training observations are equi-distant from the maximal margin hyperplane and lie around the dashed lines (showing the width of the margin).
- These three observations are known as *support vectors* (they are vectors in higher dimensions).
- They *support* the maximal margin hyperplane in the sense that if these points were moved slightly, then the maximal margin hyperplane would move as well.
- The maximal margin hyperplane depends directly on only a small subset of the observations!
- A classifier based on a separating hyperplane will necessarily perfectly classify all of the training observations. This can lead to sensitivity to individual observations.

# Support Vectors

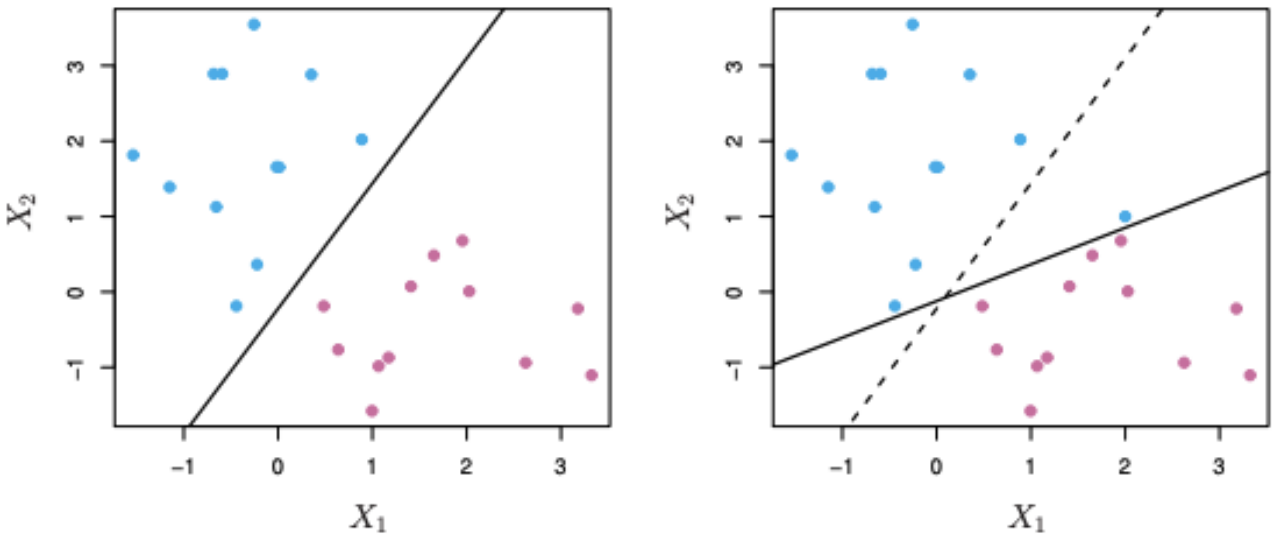



Figure 9.5 in ISLR. Left: Two classes with maximal margin hyperplane. Right: Additional blue observation was added, leading in shift in the maximal margin hyperplane (solid line). The previous one is shown as dashed line.

# Support Vector Classifier

- So far, we have assumed that a perfectly separating hyperplane exists.
- It turns out that it is possible to extend the concept of separating hyperplanes to develop a hyperplane that "almost" separates the classes.
- This classification to the non-separable case is called a *support vector classifier*.

# Non-Separable

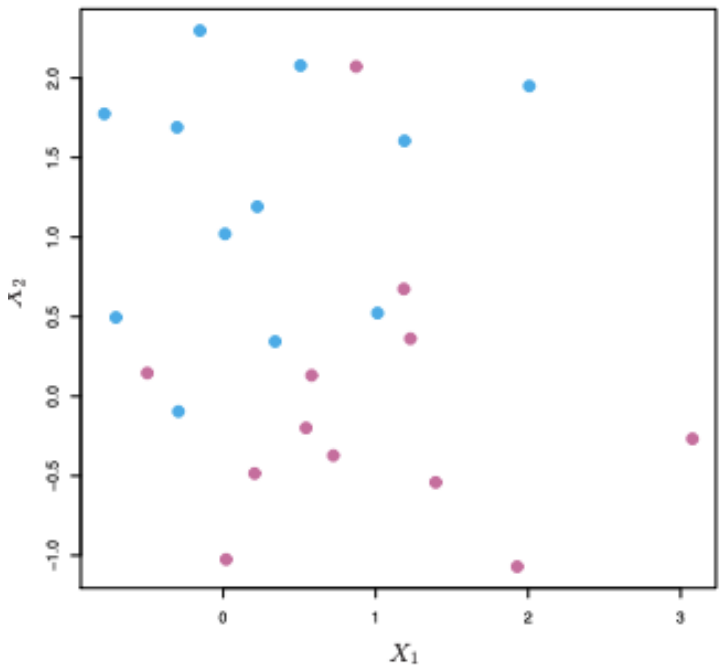



Figure 9.4 in ISLR. In this case, the classes are not separable by a hyperplane, so can't use the maximal margin classifier.

# Support Vector Classifier

- Idea: It might be worthwhile to misclassify a few training observations to do a better job in classifying the remaining obsevations.
- The Support Vector Classifier (or soft margin classifier) does exactly this.
- We are not seeking the largest margin so that every observation is not only on the correct side of the hyperplane, but also on the correct side of the margin ...
- We allow some observations to be (i) on the incorrect side of the margin and (ii) on the incorrect side of the hyperplane.
- Observations on the wrong side of the hyperplane correspond to training observations that are misclassified by the support vector classifier.

# Support Vector Classifier

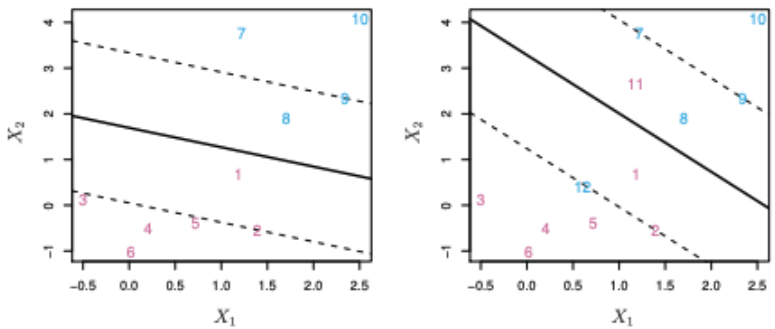



Figure 9.6 in ISLR. Hyperplane as solid line, margins as dashed lines.

# Support Vector Classifier: Left Panel (Previous Picture)

Purple observations:

- 3, 4, 5, 6 are on the correct side of the margin.
- Observation 2 is on the margin.
- Observation 1 is on the wrong side of the margin.

Blue observations:

- 7, 10 are on the correct side of the margin.
- 9 is on the mragin.
- 8 is on the wrong side of the margin.

No observation on the wrong side of the hyperplane.

# Support Vector Classifier: Right Panel (Previous Picture)

- Same as left panel, but observations 11 and 12 are on the wrong side of the hyperplane (and on the wrong side of the margin).

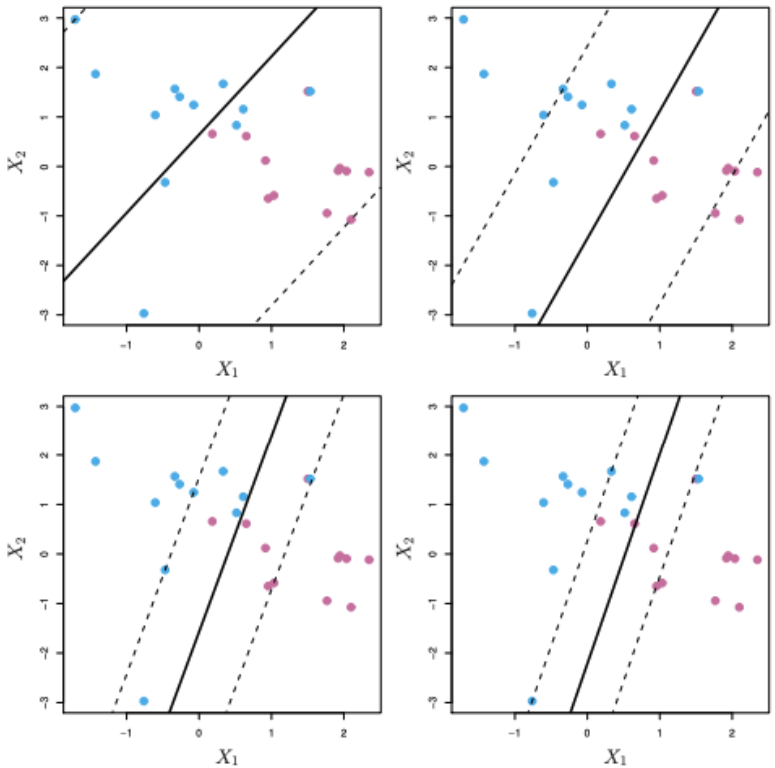
# Support Vector Classifier

- Classify a test observation depending on which side of a hyperplane it lies.
- The hyperplane is chosen to correctly separate *most* of the training observations into the two classes, but it may misclassify a few observations.
- Specifically, allow for a "budget" (say C) for the amount by which we can violate the margin (and to the hyperplane). This can be treated as a tuning parameter - and chosen via cross validation.
- C also has a bias-variance trade off: If C is small, we see narrow margins that are rarely violated, so may have low bias, but may have high variance. If C is large, we allow more violations, so may have higher bias but lower variance.

# Aside: Budget - Support Vector Classifier

- What specifically is this budged? We constrain $\sum_{i=1}^{n} \epsilon_i \leq C$ where $\epsilon_i > 0$.
- If $\epsilon = 0$, then then the $i$th observation is on the correct side of the margin (see the maximal margin classifier).
- If $\epsilon > 0$, then the observation is on the wrong side of the margin (and it has violated the margin).
- If $\epsilon > 1$, then it is on the wrong side of the hyperplane.
- If $C > 0$, no more than $C$ observations can be on the wrong side of the hyperplane, because if an observation is on the wrong side of the hyperplane, then $\epsilon_i > 1$.

# Aside: Budget - Support Vector Classifier




ISLR figure 9.7: Largest value of C was used in top left panel - large tolerance for observations to be on the wrong side of the margin (large margin). Smaller values of C were used in top right, lower left, lower right panel. As C decreases, margin narrows (less tolerance for observations being on the wrong side of the margin).

# Support Vector Classifier

- Property of the optimization procedure: It turns out that only observations that either lie on the margin or that violate the margin will affect the hyperplane (i.e. the support vectors).
- This is why it is called the support vector classifier.
- But the support vector classifier only seeks a linear boundary, and therefore would do poorly in situations like the following:

# Support Vector Classifier

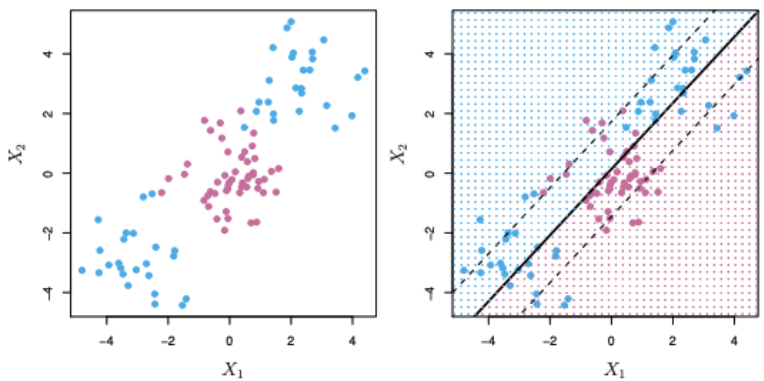


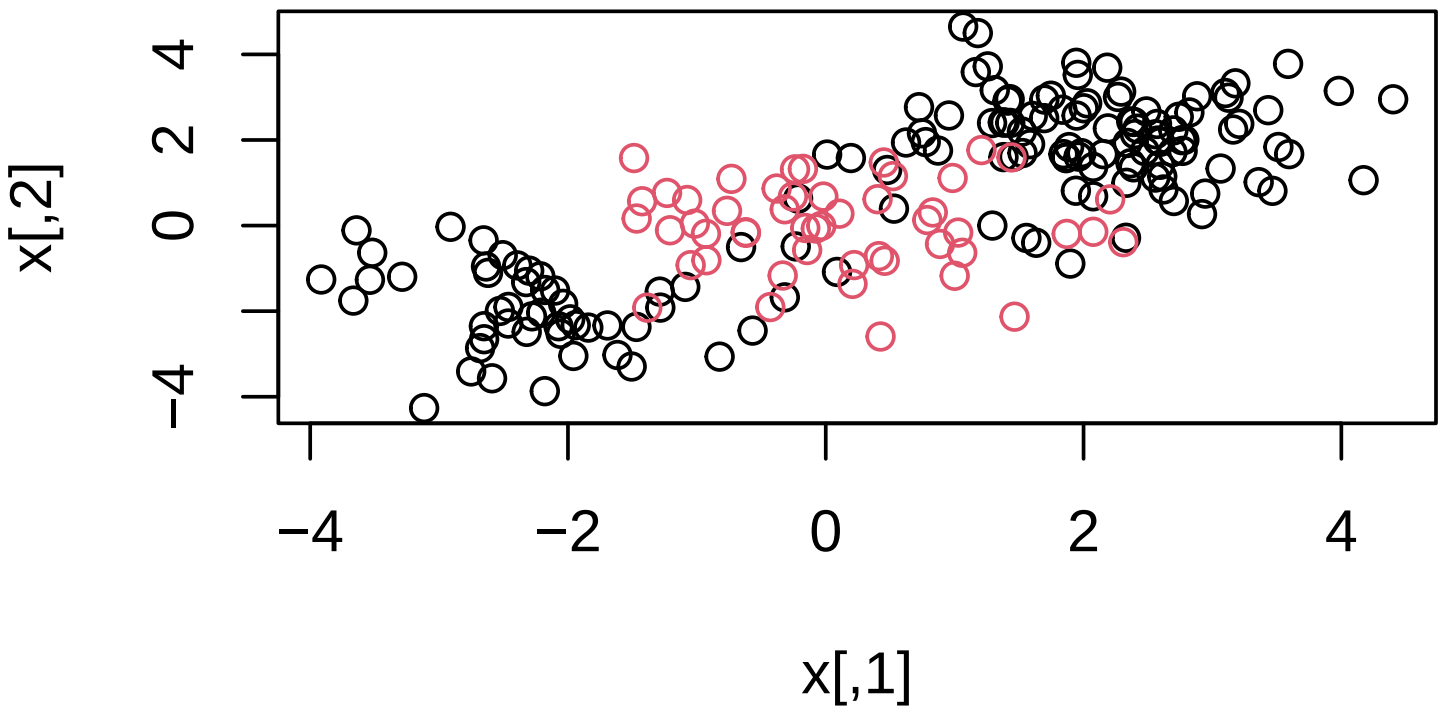

Figure 9.8 in ISLR. The observatins fall into two classes, with a non-linear boundary between them. Right: The support vector seeks a linear boundary, and therefore performs poorly.

# Support Vector Machine

- The support vector machine (SVM) is an extension of the support vector that allows for non-linear decision boundaries. This is done by "kernels" - if you are interested in learning more, you can read about them in chapter 9 of ISLR!
- Can fit using the `svm()` function in R. Using `tune()` to automatically perform cross-validation.

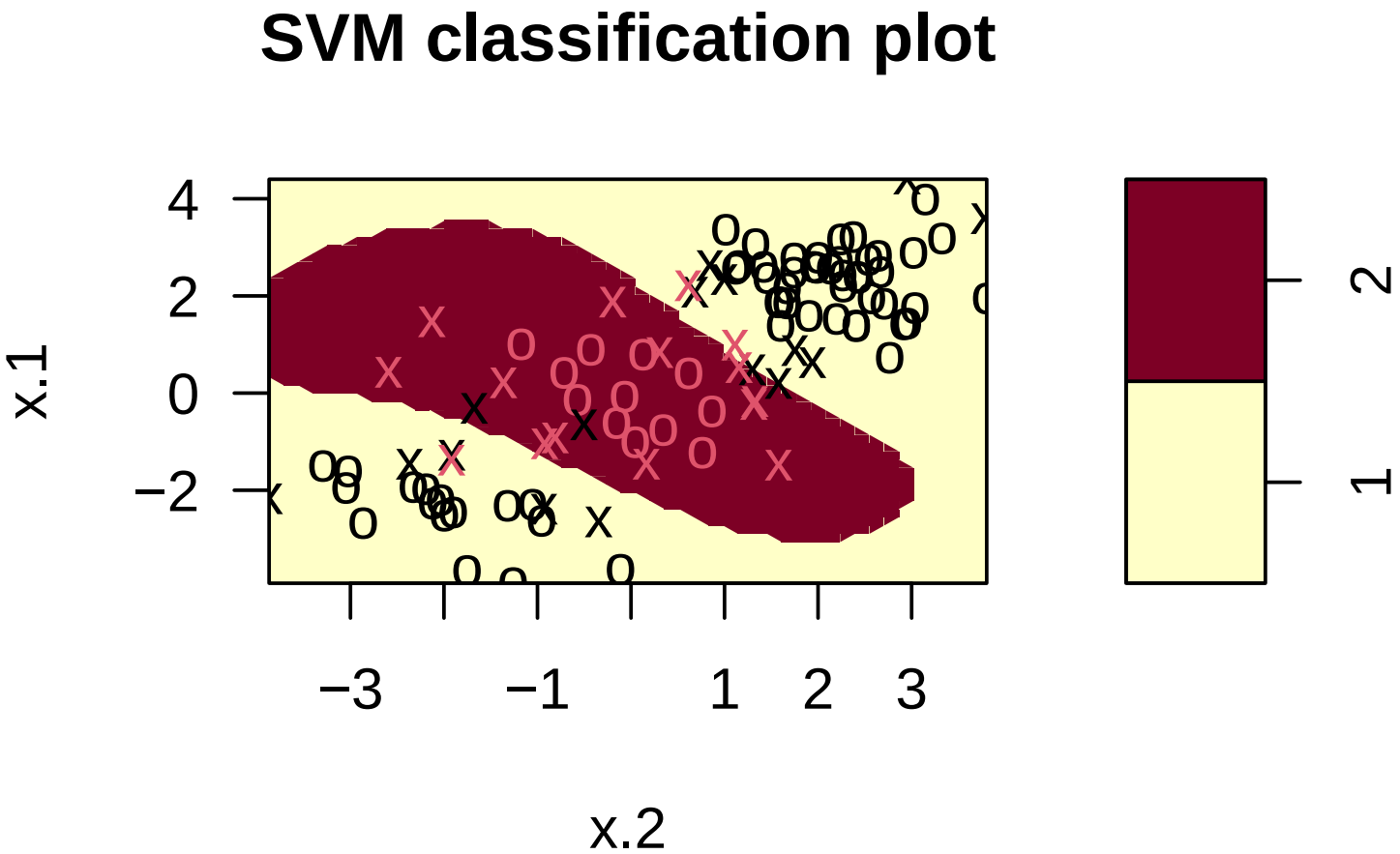
# Support Vector Machine in R

```
set.seed (1)
library(e1071)
set.seed (1)
x <- matrix(rnorm (200 * 2), ncol = 2)
x[1:100, ] <- x[1:100, ] + 2
x[101:150, ] <- x[101:150, ] - 2
y <- c( rep (1, 150), rep (2, 50))
dat <- data.frame (x = x, y = as.factor(y))
plot (x, col = y)
```



# Support Vector Machine in R

```
train <- sample (200, 100)
svmfit <- svm ( y ~ ., data = dat[train , ], kernel = "radial", gamma = 1, cost = 1)
plot (svmfit , dat[train , ])
```




- It is possible to use `tune()` to perform cross validation, see lab 9.6 in ISLR.

# Support Vector Machine in R

- Predict cancer subtype using gene expression measurement. Data set of gene expression measurements for 2,308 genes.

```
library(ISLR)
# fit on training data
dat <- data.frame (x = Khan$xtrain ,y = as.factor(Khan$ytrain))
out <-  svm(y ~ ., data = dat, kernel = "linear", cost = 10)

# predict on test data
dat.te <- data.frame(x = Khan$xtest ,y = as.factor (Khan$ytest))
pred.te <- predict(out , newdata = dat.te)
table(pred.te , dat.te$y)
```

```
pred.te 1 2 3 4
      1 3 0 0 0
      2 0 6 2 0
      3 0 0 4 0
      4 0 0 0 5
```