Support vector machines (SVM）

A support vector machine (SVM) is a supervised learning model that uses associated algorithms to analyze data and recognize patterns. It is mostly used in classification with binary classifications settings. It can also be extended to multiple cases. After giving SVM models cases of classified training data for each label, they‘re able to categorize new data.

https://en.wikipedia.org/wiki/Support-vector_machine

Contents

Terminology

Hyperplane: A hyperplane is a subspace with one less dimension than that the ambient space.

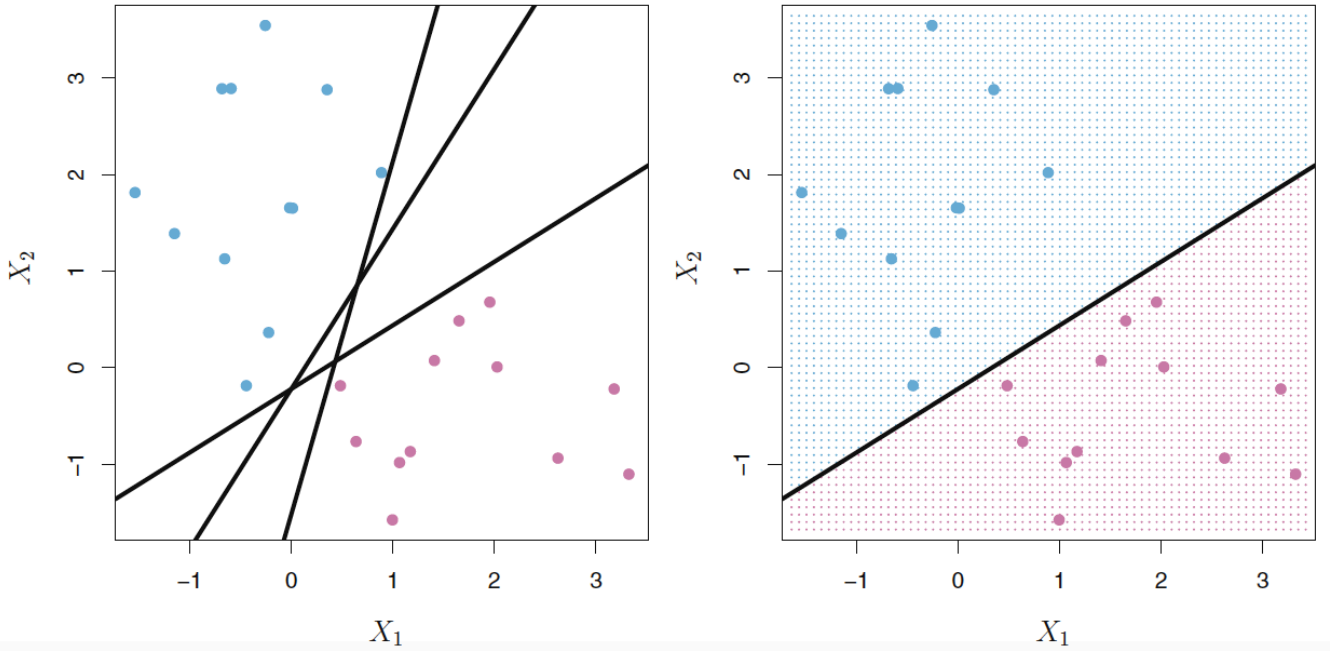
Each separating hyperplane is a classifier: a test observation is mapped to the plane with its features and then assigned to a class according to which side of the hyperplane it is located.

Margin: the shortest distance from our observations to a hyperplane. **How to determine the separating hyperplane?**

Reference : https://en.wikipedia.org/wiki/Hyperplane

Some intuition

Our goal is to **define a classifier** using these points that will correctly classify a new observation.

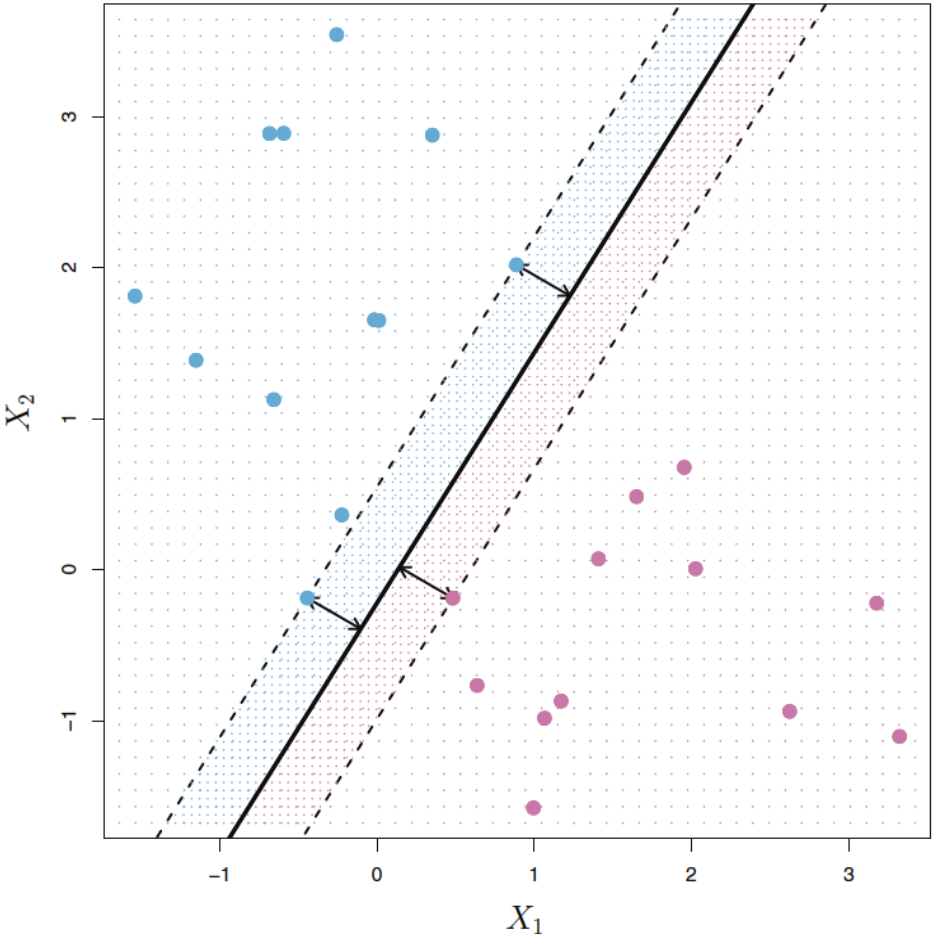


The blue and red dots represent two classes. The axes represent their features. All lines in the left picture are hyperplanes. As you can see, there are infinitely many hyperplanes.

Source: *An Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani; page 340

The Maximal Margin Classifier

- The maximal margin hyperplane is the separating hyperplane having the largest margin.
- We choose **the maximal margin classifier** in the hope of it will also have a large margin on the test data and hence will classify the test data correctly.
- Although the maximal margin classifier is helpful, it can also lead to overfitting problem.

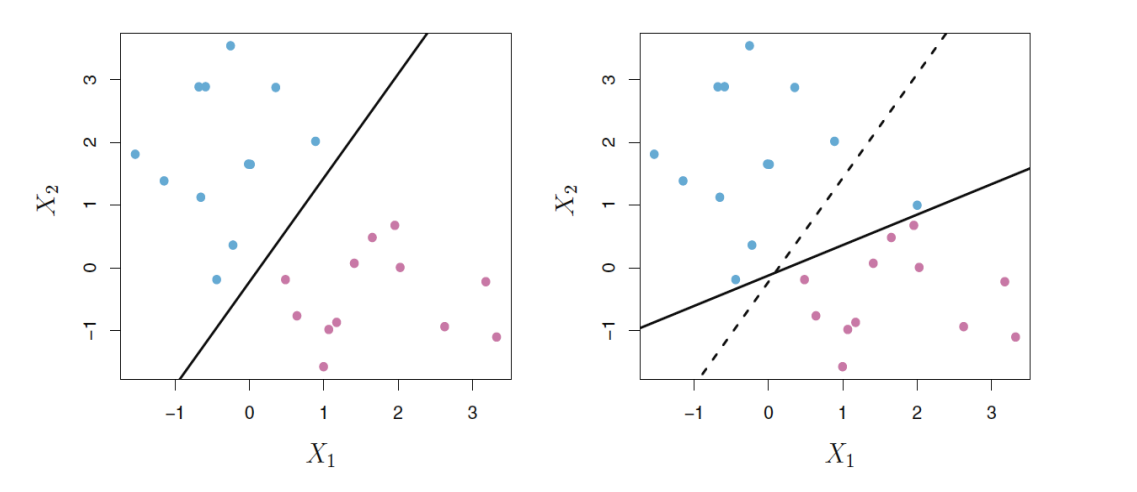


Source: *An Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani; page 342

The Maximal Margin Classifier and Support Vector Classifiers

- Just like any machine learning algorithms, we will consider bias-variance tradeoff. Here I will use two-dimensional data points to illustrate. If we used a threshold that is very sensitive to the training data (although is has low bias), it will perform badly with a new data. However, if we pick a threshold that allows some misclassifications (with high bias), it will perform better in latter test (low variance).

The Maximal Margin Classifier and Support Vector Classifiers



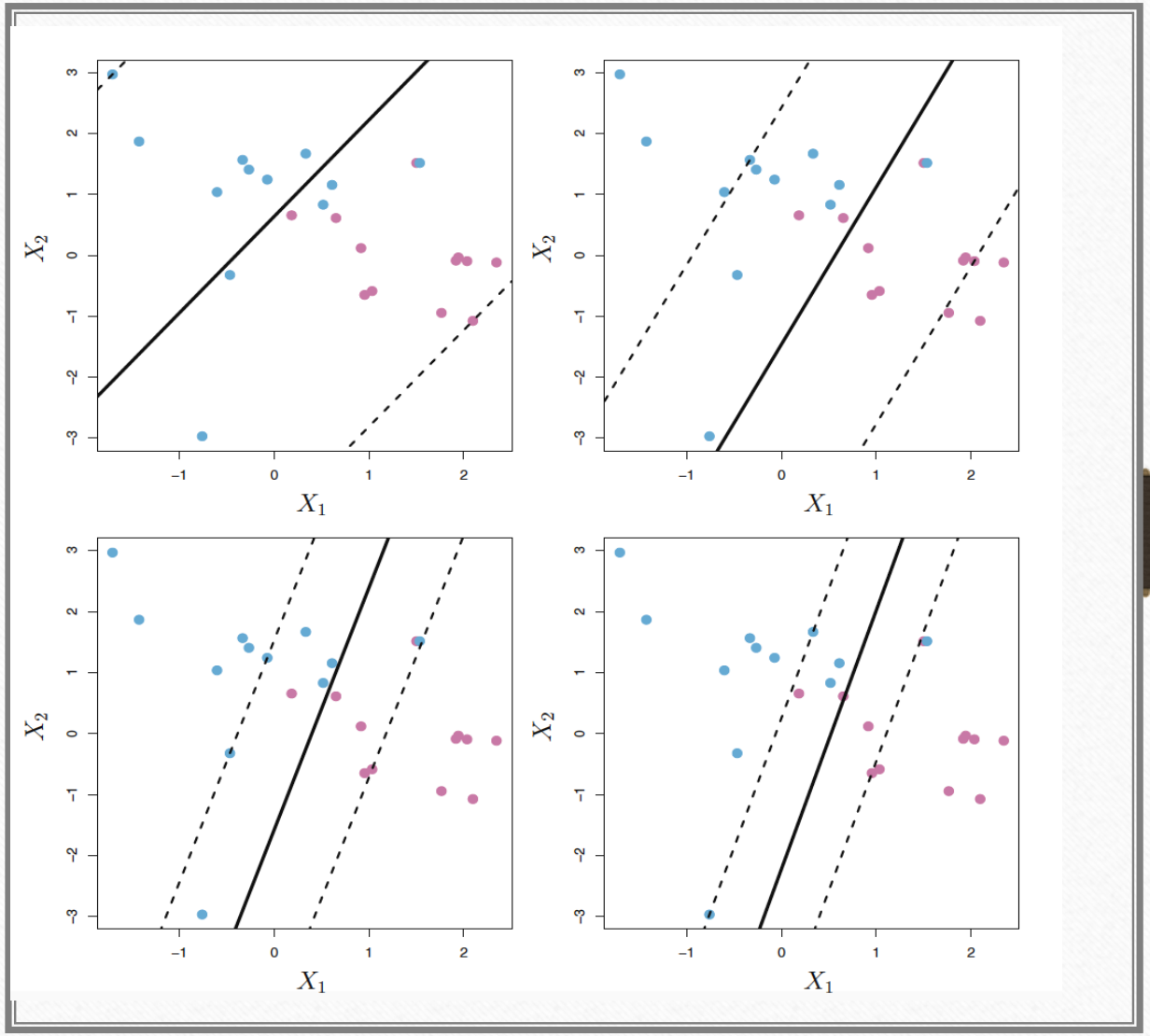
Source: *An Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani; page 345

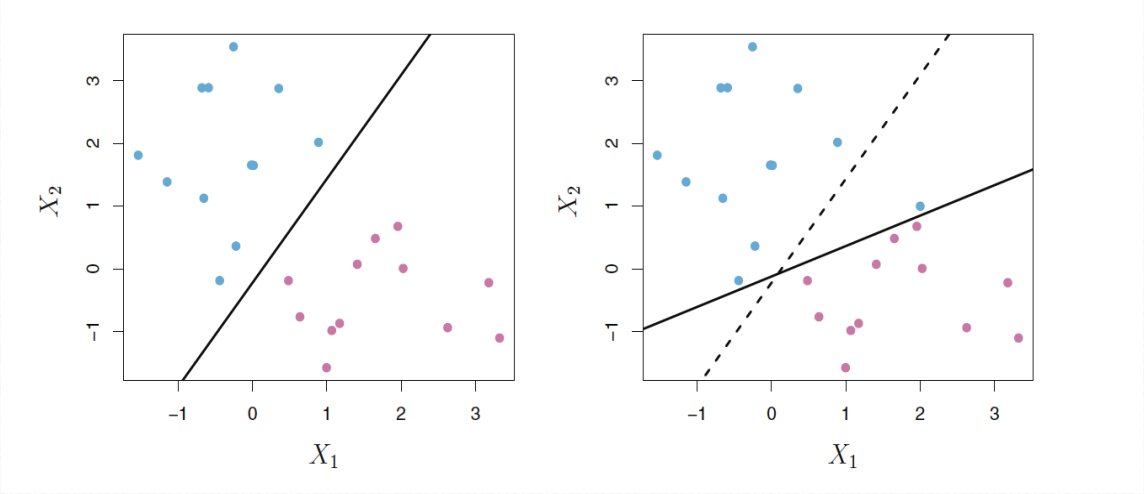
- The separating line changed dramatically after the extra blue point was added in the right picture.
- That's how **Support Vector Classifiers** comes in for help**.** It is also called Soft Margin Classifier since it allows some observations to be in the margin or the wrong side of the hyperplane)

The Support Vector Classifier

We use Cross Validation to determine that how many misclassifications are allowed in order to get a better long- run result (different tuning parameters result in different margins)

Source: *An Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani; page 348





Now the additional blue points will not affect our classifier since we allow misclassifications.

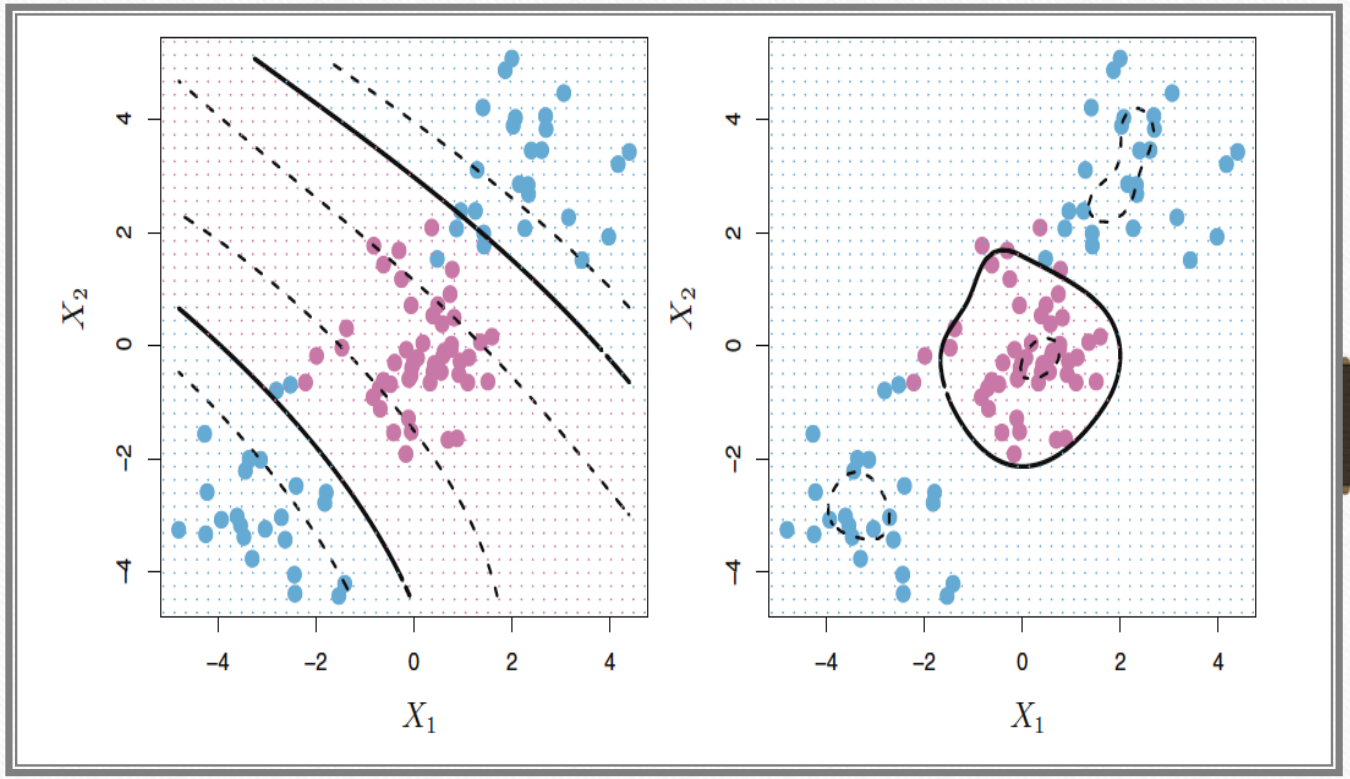
But does this solve all the problems?

The SVM algorithm could ignore outliers and find the hyperplane that has the maximum margin. Hence, we say it has robustness. It can handle outliers.

One thing to notice is that an observation that lies strictly on the correct side of the margin does not affect the support vector classifier. Observations that lie on the margin, or on the wrong side of the margin for the right class, are known as support vectors.
These observations will affect the support vector classifier.

Suppose now our data has overlapping points

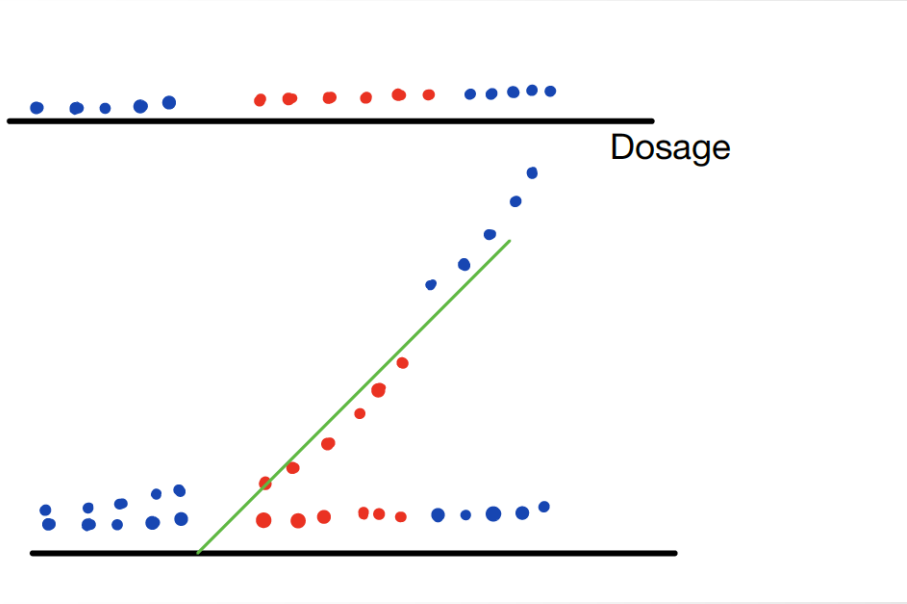
- Here the red points are in the middle of the blue points. It seems that now **no matter** where we put the classifier, we will make a lot of misclassifications as can be seen from the picture.
- It is high time we talked about **support vector machines**



Source: *An Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani; page 353

Support Vector Machines

- To get the intuition behind the idea of SVMS. Let us start with one-dimension data.
- Here I have drawn a set of points. The x axis represents the dosage.
- After we convert the dosage to squared terms, we can now apply support vector classifier



Kernel Trick

- Before introducing how exactly does SVMS work, we need to learn a new term: The SVM kernel. In order to convert non-separable problem into separable problem just like what we did in last slide.
- The SVM kernel is a function that takes low dimensional space and transforms it to a higher dimensional space It is useful in non-linear separation problem.
- For example, the transformation in our last example has **degree** 2.

Kernel Function

- Formal definition:

  If we have our one pair of input data x and y $\in X$, and a mapping:

  $\Phi: X \xrightarrow[n>\dim(X)]{} \mathbb{R}^n$, then our kernel function is $K(x,y) = < \Phi(x), \Phi(y) >$

- Kernel corresponds to an inner product in a feature space based on mapping Φ
- The function allows us to calculate a relationship between every pair of points to find their relationships in higher dimension instead of actually transferring the data to higher dimension.
- There are also many 3D visualization videos online explaining this idea

The relationships in this transformation is used to find a support vector classifier.

- We will see how kernel functions with two examples.
- One example of SVM Kernel function is **Polynomial Kernel** $K(x, y) = (x^{\mathrm{T}}y + c)^d$

- Another example is called radical kernel $K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$

Source: https://en.wikipedia.org/wiki/Radial_basis_function_kernel

Source: https://en.wikipedia.org/wiki/Polynomial_kernel

Polynomial Kernel

- **Kernel function with d =2**

$$K(x, y) = \left(\sum_{i=1}^{n} x_i y_i + c\right)^2 = \sum_{i=1}^{n} \left(x_i^2\right)\left(y_i^2\right) + \sum_{i=2}^{n}\sum_{j=1}^{i-1} \left(\sqrt{2}x_i x_j\right)\left(\sqrt{2}y_i y_j\right) + \sum_{i=1}^{n} \left(\sqrt{2c}x_i\right)\left(\sqrt{2c}y_i\right) + c^2$$

With mapping:

$$\varphi(x) = \langle x_n^2, \ldots, x_1^2, \sqrt{2}x_n x_{n-1}, \ldots, \sqrt{2}x_n x_1, \sqrt{2}x_{n-1}x_{n-2}, \ldots, \sqrt{2}x_{n-1}x_1, \ldots, \sqrt{2}x_2 x_1, \sqrt{2c}x_n, \ldots, \sqrt{2c}x_1, c\rangle$$

- ***x*** **and** ***y*** **are vectors in the input space. C and d are determined by cross validation.**
- **The kernel function is essentially the dot product, which gives a high-dimension coordinates of the data.**
- **The most common degree is** $d = 2$ **(quadratic), higher degree might cause overfitting problem.**

Source: https://en.wikipedia.org/wiki/Polynomial_kernel

Radial Kernel

- The radical kernel takes the form:

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2).$$

Or the radical basis function (RBF)

- Coefficient is determined by cross validation
- Radical kernel could work with infinite dimensions.

Source: *An Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani; page 352

How does Radical Kernel work

- Radical kernel works in a similar way to radical kernel, except that the degree of the transformation could be raised to infinity.
- Radical kernel also produces a **Dot Product** that has coordinates for an infinite dimension
- One advantage of radical kernel is that it saves time for computing.
- Remember that in polynomial kernel, we must calculate all of K(x,y), which would be horrible if we have infinite dimension data !

In summary…

- Support Vector Machines work by giving the high-dimensional relationships
- By doing so, SVMS becomes more efficient because it can avoid transferring data to high dimensions
- The mathematical methodology behind the operation of dot product is very complex.

SVMs with More than Two Classes

So far, our concepts of hyperplanes and methods of using support vector classifier seem to only work in the binary classification setting. But as we previously mentioned, SVMS can be extended to multiple classes. There are two main approaches **one-versus-one** and **one-versus-all** approaches. We will talk about one-versus-one approach here.

One-Versus-One Classification

- Assume we have n > 2 classes. A one-versus-one approach trains all pairs of classes, which gives us $\binom{n}{2}$ classifiers. During the classification computing processes each classifier predicts one class. The class which has been predicted most is the answer.
- For example, we have A,B,C classes and AB, AC, BC classifiers with subset of data.
- If: AB and AC predicts A and BC predicts B. Then we will assign it to A since it has been predicted most.
- However, this is computationally expensive.

Source: *An Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani; page 355

For more mathematics details behind the algorithms, check out chapter 9 in *An Introduction to Statistical Learning*

$$\underset{\beta_0,\beta_{11},\beta_{12}\ldots,\beta_{p1},\beta_{p2},\epsilon_1,\ldots,\epsilon_n,M}{\text{maximize}} M$$

$$\text{subject to } y_i\left(\beta_0+\sum_{j=1}^{p}\beta_{j1}x_{ij}+\sum_{j=1}^{p}\beta_{j2}x_{ij}^2\right)\geq M(1-\epsilon_i),$$

$$\sum_{i=1}^{n}\epsilon_i\leq C,\quad \epsilon_i\geq 0,\quad \sum_{j=1}^{p}\sum_{k=1}^{2}\beta_{jk}^2=1.$$

Source: *An Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani; page 350

Support Vector Machines using R

With iris data set

The SVM() function in R

- We use the e1071 library in R to explore the support vector classifier

  and the SVM. Alternatively we could use the LiblineaR library, which is used for large problems.

- If the response contains multiple levels, then the svm() function will perform multi-class classification using the one-versus-one approach
- If the response is numerical, the svm() will perform support vector regression
- In our example, we will use categorical variable with 3 levels as response variable

Source: https://www.rdocumentation.org/packages/e1071/versions/1.7-3/topics/svm

Data Loading

```
# Load the dataset package to get iris data
library(datasets)
# load the data
data(iris)
summary(iris)
```

```
 Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

Build the Model

#install.packages('e1071')

library(e1071)

library(tidyverse)

library(caret)

svm() function will build and train a support vector machine model

```
# S3 method for formula
svm(formula, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
# S3 method for default
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncol(x),
coef0 = 0, cost = 1, nu = 0.5,
class.weights = NULL, cachesize = 40, tolerance = 0.001, epsilon = 0.1,
shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,
..., subset, na.action = na.omit)
```

Source: https://www.rdocumentation.org/packages/e1071/versions/1.7-3/topics/svm

SVM( )-Argument

- Source: https://www.rdocumentation.org/packages/e1071/versions/1.7-3/topics/svm

kernel the kernel used in training and predicting. You might consider changing some of the following parameters, depending on the kernel type.

linear: $u'v$

polynomial: $(\gamma u'v + coef0)^{degree}$

radial basis: $e^(-\gamma|u-v|^2)$

sigmoid: $tanh(\gamma u'v + coef0)$

formula a symbolic description of the model to be fit.

data an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'svm' is called from.

x a data matrix, a vector, or a sparse matrix (object of class `Matrix` provided by the Matrix package, or of class `matrix.csr` provided by the SparseM package, or of class `simple_triplet_matrix` provided by the slam package).

y a response vector with one label for each row/component of `x` . Can be either a factor (for classification tasks) or a numeric vector (for regression).

scale A logical vector indicating the variables to be scaled. If `scale` is of length 1, the value is recycled as many times as needed. Per default, data are scaled internally (both `x` and `y` variables) to zero mean and unit variance. The center and scale values are returned and used for later predictions.

Training the model

```
# This  code  for  the use  of generating the same result future
set.seed(100)
#  We partition the data in to training dataset and testing dataset
training.samples <-   iris$Species %>% createDataPartition(p = 0.8, list = FALSE)
train.data <- iris[training.samples, ]
test.data <- iris[-training.samples, ]
# This is the code used to
model <- svm(Species ~ ., data=train.data)
summary(model)
```

Result

```
Call:
svm(formula = Species ~ ., data = train.data)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  46

 ( 9 19 18 )


Number of Classes:  3

Levels:
 setosa versicolor virginica
```

Predictions

Accuracy : 0.9333

Confusion Matrix

Predicted.values	setosa	verisicolor	virginica
setosa	10	0	0
verisicolor	0	9	1
virginica	0	1	9

```
predicted.values <- predict(model,test.data[1:4])

    table(predicted.values,test.data[,5])
```

	Class: setosa	Class: versicolor	Class: versico
Sensitivity	1.0000	0.9000	0.9000
Specificity	1.0000	0.9500	0.9500

Tuning

```
# We use gamma from 0.001 to 2 and costs 0.01 to 100

# Set kernel to be radical kernel

result <-
tune(svm,train.x=train.data[1:4],train.y=train.data[,5],kernel='radial', ranges=list(cost=10^(-3:3), gamma=c(0.001:2)))

summary(result)
```

The best performance
Cost =1000
Gamma = 0.001

We can now use our parameters with best performance

```
svm.tune <- svm(Species ~ ., data=train.data, kernel="radial", cost=1000, gamma=0.001)

summary(svm.tune)

# Alternatively, we could use other parameters to see if the result has any difference
```

Predictions

```
# We use our tuned model to predict our test data
# The predicted value is saved in tuned.predicted.values
tuned.predicted.values <- predict(svm.tune, test.data[1:4])
# table() function computes the confusion matrix
table(tuned.predicted.values ,test.data[,5])
```

Result

Bam

Predicted.values	setosa	verisicolor	virginica
setosa verisicolor	10 0	0 9	0 0
virginica	0	1	10

We got a slightly better result. The improvement will be bigger if we have a larger data set.
However, we still expect to get a better result after we tuned our model

Accuracy : 0.9667

	Class: setosa	Class: versicolor	Class: versico
Sensitivity	1.0000	0.9000	1.0000
Specificity	1.0000	1.0000	0.9500

Logistic Regression

Versus

Support Vector Machines

Comparing logistic regression to support vector machines.

We know that Logistic Regression is used widely to examine the relationship binary response variable and. A bunch of predictors. Is there any relationship between SVMS and it?

The logistic regression solved classification problems by modeling the probability of the classes while support vector machines are trying to find an optimal hyperplane. It also uses the idea of kernel function. Support vector machines seems significantly different from logistic regression.

However, after rewriting the criterion for fitting the support vector classifier, we will be able to see the connections between them.

In Support Vector Machines

The textbooks gives the formula that takes "Loss + Penalty" form:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{minimize}} \left\{ \sum_{i=1}^{n} \max\left[0, 1 - y_i f(x_i)\right] + \lambda \sum_{j=1}^{p} \beta_j^2 \right\},$$

This takes the the "Loss + Penalty" form

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{minimize}} \left\{ L(\mathbf{X}, \mathbf{y}, \beta) + \lambda P(\beta) \right\}.$$

L(X, y, p(β)) is some loss function and P(β) is a penalty function, with **hinge loss**:

$$L(\mathbf{X}, \mathbf{y}, \beta) = \sum_{i=1}^{n} \max\left[0, 1 - y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})\right].$$

Source: *An Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani; page 356

In Logistics Regression

- Recall that for logistic regression, we could have a log-likelihood plus a ridge penalty.

$$L(\mathbf{X}, \mathbf{y}, \beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$$

The textbook suggests that when we have clearly-separated classes,

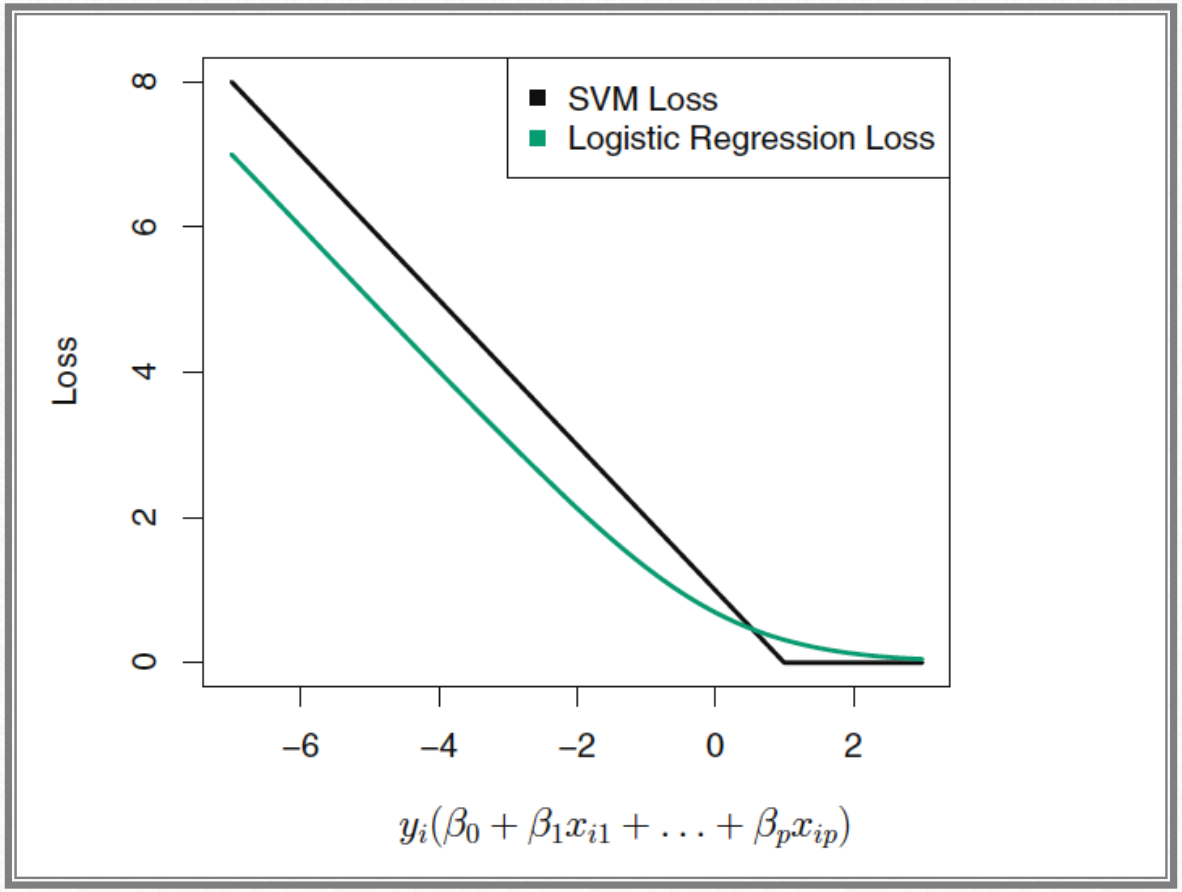
SVMs is a better choice than logistic regression; when encounter with

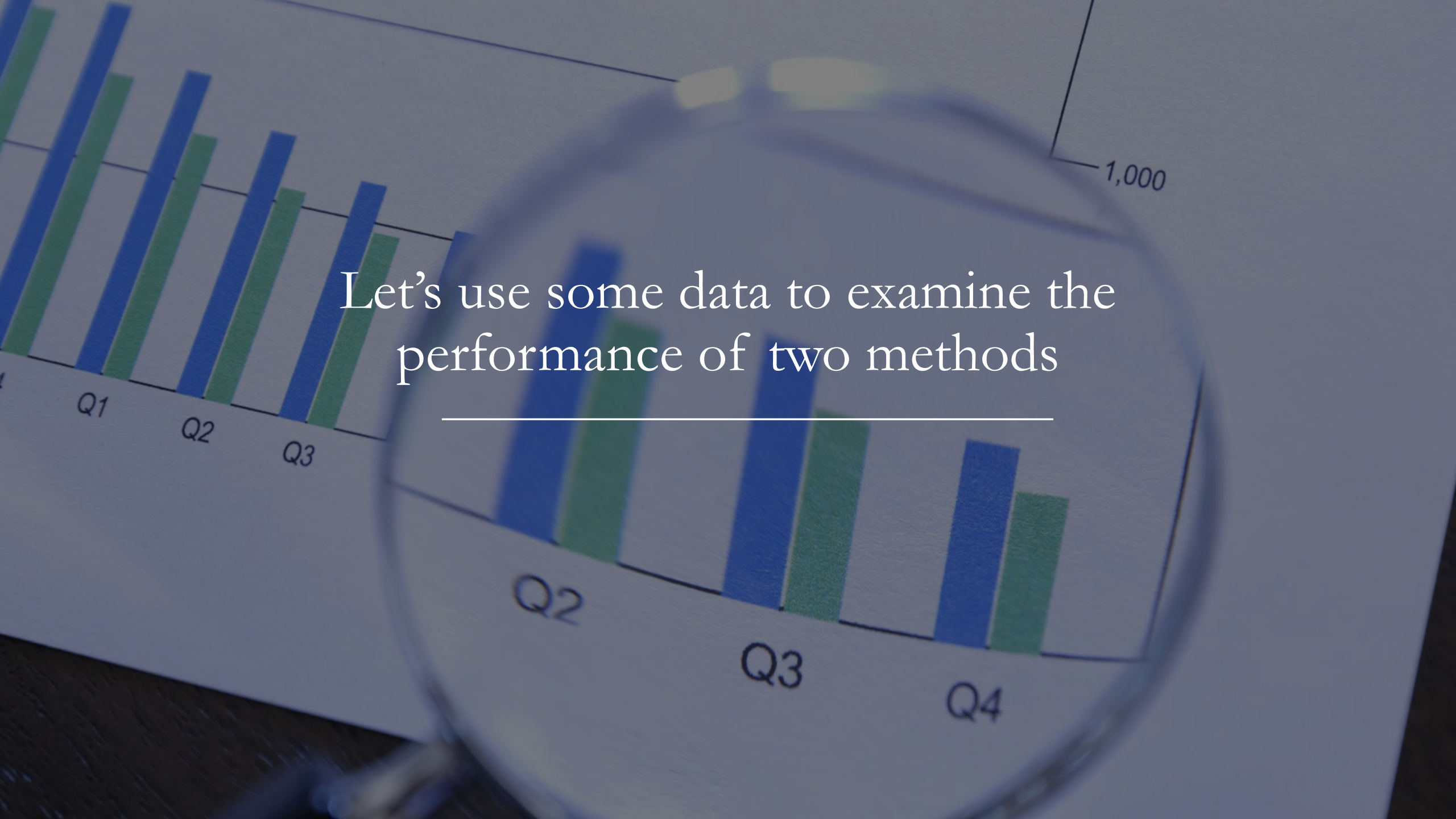
overlapping data, logistic regression is more preferred

Source: *An Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani; page 356

The function looks the same. The textbook put them together as a function of y_i ($\beta 0 + \beta 1$*x1+…) in a graph showing on the right. From the picture, we can see some similarity

Source: *An Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani; page 358





Let's use some data to examine the performance of two methods

Introduction to the data we use

To compare between logistics regression, we will use the titanic datasets.
You can find it on here:
https://www.kaggle.com/c/titanic
The data contains some information about the passengers and the goal is to create a model that predicts which passengers survived the Titanic shipwreck.
These data sets are well-prepared for a machine learning context including a training sample, a testing sample, and two additional data sets that can be used for deeper machine learning analysis

Source: https://cran.r-project.org/web/packages/titanic/titanic.pdf

Let's use head() function to see how our data looks like

Id	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1000	C123	S
7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
11	1	3	Sandstrom, Miss. Marguerite Rut	female	4	1	1	PP 9549	16.7000	G6	S
12	1	1	Bonnell, Miss. Elizabeth	female	58	0	0	113783	26.5500	C103	S
22	1	2	Beesley, Mr. Lawrence	male	34	0	0	248698	13.0000	D56	S

Let's use str() function to examine our data to see how we should process the data

We have 183 obsevations. of 12 variables:
PassengerId: int 2 4 7 11 12 22 24 28 53 55 ...
Survived : int 1 1 0 1 1 1 1 0 1 0 ...
Pclass : int 1 1 1 3 1 2 1 1 1 1 ...
Name : chr "Cumings, Mrs. John Bradley (Florence Briggs Thayer)"
Sex : chr "female" "female" "male" "female" ...
Age : num 38 35 54 4 58 34 28 19 49 65 ...
SibSp : int 1 1 0 1 0 0 0 3 1 0 ...
Parch : int 0 0 0 1 0 0 0 2 0 1 ...
Ticket : chr "PC 17599" "113803" "17463" "PP 9549" ...
Fare : num 71.3 53.1 51.9 16.7 26.6 ...
Cabin : chr "C85" "C123" "E46" "G6" ...
Embarked : chr "C" "S" "S" "S" ...

Data Processing

- Note :
- SibSp : Number of Siblings Spouses Aboard
- Parch : Number of Parents/Children Aboard
- Ticket : Ticket Number
- Fare Passenger Fare

```
install.packages('titanic')
library(titanic)
library(dplyr)
library(caret)
raw.data <- titanic_train
# NA consistency
raw.data[raw.data==""] <- NA
# check whether there is missing values
colSums(is.na(raw.data))
# Get rid of NA values
raw.data <- na.omit(raw.data)
#Get rid of the variables that are not very useful in our regression
new.data <- subset(raw.data, select = c(2,3,5,6,7,8,10,12))
# Factor our prediction variable
new.data$Survived <- as.factor(new.data$Survived)
#split into training set and test set
split <- new.data$Survived %>%
createDataPartition(p = 0.75, list = FALSE)
train <- new.data[split,]
test <- new.data[-split,]
```

Logistic Regression

```
r.model <- glm(Survived ~.,family=binomial(link='logit'),data=train)

summary(r.model)

# Make predictions on testing set

pred <- r.model %>% predict(test,type = "response")

# Set the threshold at p = 0.5

pred <- ifelse(pred > 0.5, 1, 0)

confusionMatrix(as.factor(pred),test$Survived, positive = "1")
```

Reference: https://www.r-bloggers.com/how-to-perform-a-logistic-regression-in-r/

Result

	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	5.398210	1.436177	3.759	0.000171	***
Pclass	-0.673421	0.513244	-1.312	0.189490	
Sexmale	-2.920956	0.574155	-5.087	3.63e-07	***
Age	-0.052566	0.018474	-2.845	0.004436	**
SibSp	0.234399	0.470663	0.498	0.618470	
Parch	-0.511363	0.393086	-1.301	0.193295	
Fare	0.002770	0.003384	0.819	0.413029	
EmbarkedQ	-1.841451	2.052932	-0.897	0.369726	
EmbarkedS	-0.072195	0.528745	-0.137	0.891395	

#Estimate of the coefficients
#Standard Error of the coefficients
#Z value and P value for the estimates

AIC: 136.4029

Null deviance: 174.26 on 137 degrees of freedom
Residual deviance: 118.40 on 129 degrees of freedom

SVMS

```
# The svm() package is in e1071 packge.
install.packages('e1071')
library(e1071)
s.model <- svm(Survived ~ ., data=train)
summary(s.model)
set.seed(123)
# This time we will train our model using train() in caret package
tune.model <- train(
  Survived ~., data = train, method = "svmRadial",
  trControl = trainControl("cv", number = 10),
  preProcess = c("center","scale"),
  tuneLength = 10
 )
```

Predictions

```
# This code will give us the best parameters for the model
tune.model$bestTune
# Sigma: 0.1589484 Cost: 0.5
# Check  out how our model performs on test data
predicted.classes <- tune.model %>% predict(test)
confusionMatrix(as.factor(predicted.classes),test$Survived,
positive = "1")
```

Result Interpretation

- Our formula is using all the variables to predict on “Survived “.
- SVM-Type: The argument is C-classification because our outcome variable is a factor
- SVM-Kernel: radial
- cost: 1
- Number of Support Vectors: 89
- Number of Classes: 2 (Survived or not)

Confusion Matrix and Statistics

Logistic regression

Prediction	Reference 0	1
0	8	5
1	7	25

Accuracy : 0.7333
Sensitivity : 0.8333
Specificity : 0.5333

Logistic regression performs better in accuracy and specificity

SVM

Prediction	Reference 0	1
0	4	3
1	11	27

Accuracy : 0.6889
Sensitivity : 0.9000
Specificity : 0.2667

Thank you!