Universidad Miguel Hernández

Facultad de Ciencias Sociales y Jurídicas de Elche

Grado en Estadística Empresarial

Predicción de variables cualitativas y métodos de clasificación

Alumno:
Víctor Manuel Pacheco Panadero

Tutora:
María Dolores Esteban Lefler

Curso Académico 2022/2023

Índice general

1. Palabras clave

Predicción, variables cualitativas, clasificación.

2. Resumen

En este Trabajo de Fin de Grado se comparan algunos métodos de clasificación para poder realizar predicciones. Los métodos de clasificación utilizados en este estudio son: regresión logística, análisis discriminante lineal, análisis discriminante cuadrático, Naive Bayes, K-Nearest Neighbors y Support Vector Machine. El objetivo es comprobar cuál método proporciona mejores resultados.

Para realizar dicho estudio utilizaremos el programa R, que es un lenguaje de programación libre y gratuito enfocado en el análisis estadístico.

En cada método de clasificación habrá una breve explicación de sus fundamentos, características, ventajas y desventajas. Dicha teoría irá acompañada de un ejemplo de cómo se aplicaría usando el lenguaje R. Una vez codificado se analizarán e interpretarán los resultados. Además, se compararán entre sí los métodos considerados para determinar si alguno de ellos es mejor.

Para desarrollar el ejemplo, se aplicarán los métodos de clasificación a una base de datos que contiene los rendimientos porcentuales del índice bursátil S&P 500 a lo largo de 1089 semanas. Esta base de datos proporcionará una amplia cantidad de información para evaluar el desempeño de los métodos de clasificación en un contexto financiero. Una vez que se hayan aplicado los métodos de clasificación y se hayan obtenido los resultados, se extraerán conclusiones y se identificarán aquellos métodos que han proporcionado los mejores resultados en términos de precisión y capacidad predictiva.

3. Objetivos

Los objetivos para la realización de este trabajo son los siguientes:

- Entender teóricamente y comparar los métodos de clasificación: regresión logística, análisis discriminante lineal, análisis discriminante cuadrático, Naive Bayes, K-Nearest Neighbors y Support Vector Machine.
- Saber aplicar correctamente los modelos anteriores en un conjunto de datos real.
- Saber cómo aplicar dichos métodos en R.
- Extraer conclusiones de los resultados obtenidos en R.

4 Base de datos a utilizar

Antes de empezar a resolver los métodos de clasificación en R se hará una descripción de la base de datos que se utilizará en el estudio. Para ello, comenzaremos examinando medidas descriptivas y gráficos de la base de datos `Weekly`, que forma parte de la librería `ISLR2`. Este conjunto de datos consta de rendimientos porcentuales semanales del índice bursátil S&P 500 durante 1089 semanas, desde principios de 1990 hasta finales de 2010. Para cada fecha, se han registrado los rendimientos porcentuales de cada uno de las cinco semanas anteriores, variables `Lag1` a `Lag5`.También se han registrado las variables `Volume` (el número de acciones negociadas la semana anterior, en miles de millones), `Today` (el rendimiento porcentual en la semana en cuestión) y `Direction` (si el mercado sube o baja en dicha semana). Nuestro objetivo es predecir la variable `Direction` (una variable cualitativa) usando las otras características.

Lo primero de todo es cargar la base de datos, para ello necesitaremos la librería `ISLR2`.

```
#1
library(ISLR2)
library(GGally)
attach(Weekly)
names(Weekly)
```

```
## [1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"      "Lag5"
## [7] "Volume"    "Today"     "Direction"
```

```
dim(Weekly)
```

```
## [1] 1089    9
```

```
summary(Weekly)
```

```
##       Year           Lag1                Lag2                Lag3
##  Min.   :1990   Min.   :-18.1950   Min.   :-18.1950   Min.   :-18.1950
##  1st Qu.:1995   1st Qu.: -1.1540   1st Qu.: -1.1540   1st Qu.: -1.1580
##  Median :2000   Median :  0.2410   Median :  0.2410   Median :  0.2410
##  Mean   :2000   Mean   :  0.1506   Mean   :  0.1511   Mean   :  0.1472
##  3rd Qu.:2005   3rd Qu.:  1.4050   3rd Qu.:  1.4090   3rd Qu.:  1.4090
##  Max.   :2010   Max.   : 12.0260   Max.   : 12.0260   Max.   : 12.0260
##       Lag4                Lag5              Volume            Today
##  Min.   :-18.1950   Min.   :-18.1950   Min.   :0.08747   Min.   :-18.1950
##  1st Qu.: -1.1580   1st Qu.: -1.1660   1st Qu.:0.33202   1st Qu.: -1.1540
##  Median :  0.2380   Median :  0.2340   Median :1.00268   Median :  0.2410
##  Mean   :  0.1458   Mean   :  0.1399   Mean   :1.57462   Mean   :  0.1499
##  3rd Qu.:  1.4090   3rd Qu.:  1.4050   3rd Qu.:2.05373   3rd Qu.:  1.4050
##  Max.   : 12.0260   Max.   : 12.0260   Max.   :9.32821   Max.   : 12.0260
##  Direction
##  Down:484
##  Up  :605
##
##
##
##
```

```
ggpairs(Weekly[, -9])
```

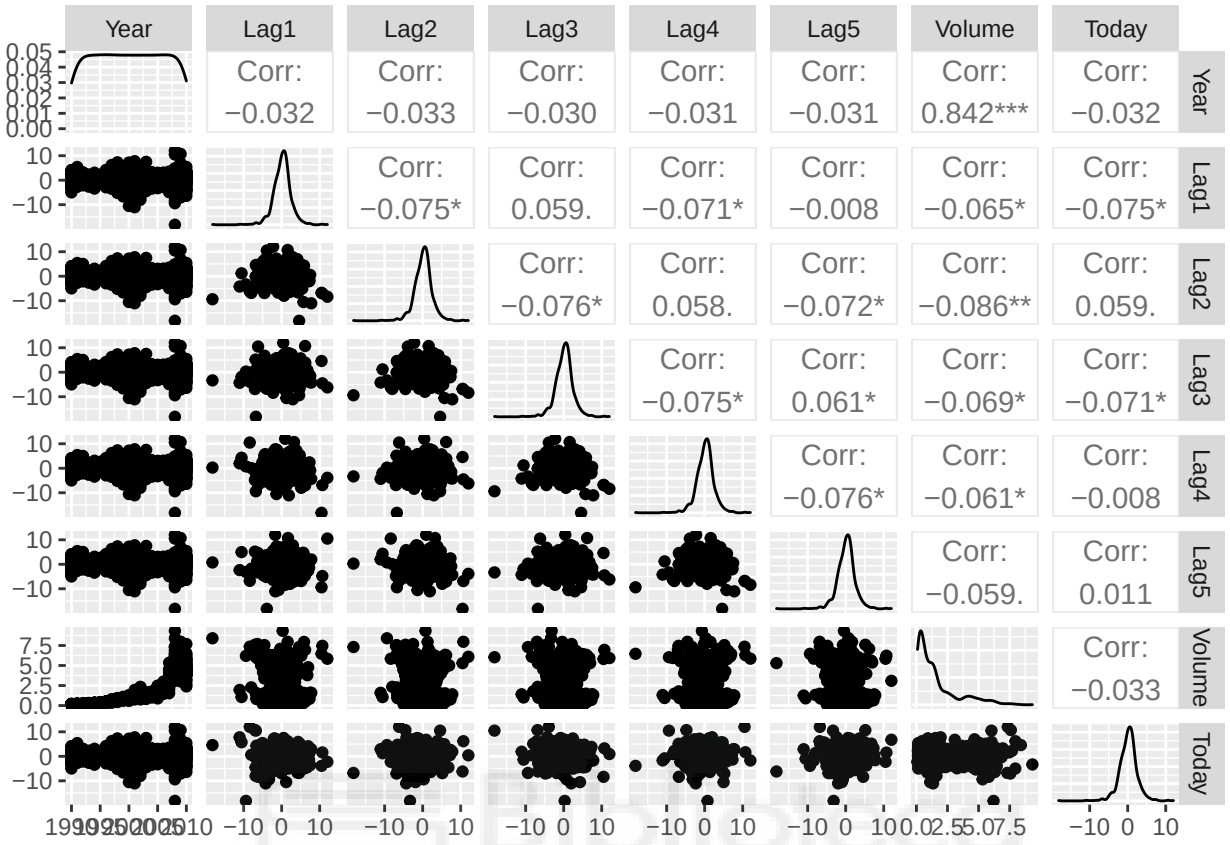


Figura 1. Gráfico de dispersión y correlaciones.

Este código (Cód.1) nos muestra el nombre de las variables de la base de datos, la dimensión, un resumen estadístico del conjunto de datos donde, si la variable es "numeric" nos proporcionará estadísticas básicas como el mínimo, el primer cuartil(Q1), la mediana(Q2), la media, el tercer cuartil(Q3) y el máximo. Además, se mostrará el número de valores faltantes (NA) en el caso que los tuviera. Si en vez de ser una variable "numeric" es una variable de clase "factor" o "character" se mostrará el conteo de cada nivel o categoría. Y por último nos mostrará una matriz de gráficos de dispersión y correlaciones, usando todas las variables menos Direction que es cualitativa, para explorar las relaciones entre múltiples variables en un conjunto de datos.

Con las salidas obtenidas (Cód.1) vemos que todas las variables son numéricas excepto la variable `Direction` que es cualitativa. Con el gráfico de dispersión (Fig.1) no parece que haya relación entre variables, excepto por la variable `Year` y `Volume` donde podemos apreciar un aumento de las negociaciones a lo largo de los años. Además, las correlaciones entre las variables Lag y los rendimientos de la semana (Today) son cercanas a cero. En otras palabras, parece haber poca correlación entre los rendimientos de la semana y los rendimientos de semanas anteriores. La única correlación sustancial es entre las variables `Year` y `Volume`.

Al graficar los datos, que están ordenados cronológicamente, vemos que los valores de la variable `Volume` aumentan con el tiempo. En otras palabras, el número promedio de acciones negociadas semanalmente aumentó de 1990 a 2010.

```
plot(Weekly$Volume)
```

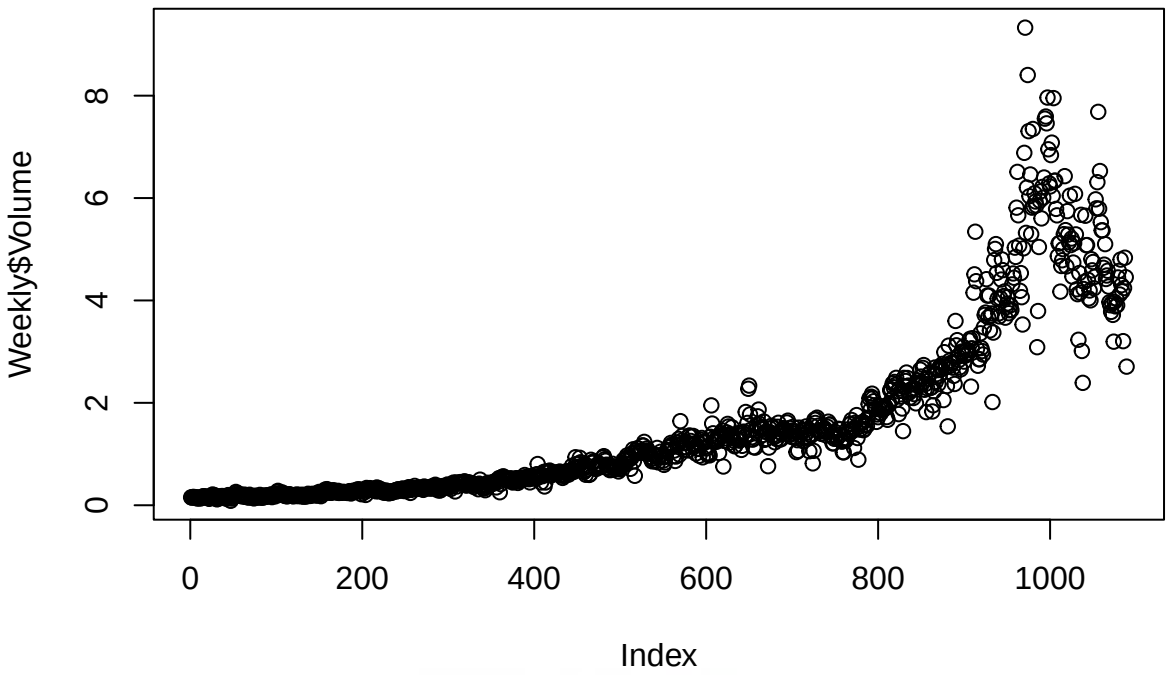


Figura 2. Gráfico de dispersión de negociaciones en función del tiempo.

```
#2
train_Weekly <- subset(Weekly, Year < 2009)
test_Weekly <- subset(Weekly, Year >= 2009)
```

Para realizar las predicciones dividiremos la base de datos en dos (Cód.2). Por un lado tendremos las observaciones pertenecientes a los años anteriores a 2009, que serán los datos de muestra donde se estimarán los parámetros del modelo, y por otro lado los datos de prueba, para los cuales utilizaremos los años 2009 y 2010 para comprobar si las predicciones han sido correctas.

5. Regresión Logística

5.1 Introducción

La regresión logística es un modelo lineal generalizado utilizado para predecir la probabilidad de ocurrencia de un evento binario (por ejemplo, sí/no, verdadero/falso) en función de variables independientes. Aunque su nombre incluye la palabra "regresión", la regresión logística se utiliza principalmente para problemas de clasificación en lugar de problemas de regresión.

5.2 Regresión Lógistica Simple

La formulación matemática de la regresión logística se basa en el modelo de regresión lineal, pero la función de regresión se transforma a través de una función logística (también conocida como función sigmoide) para obtener una probabilidad. La función logística tiene la siguiente forma:

$$f(z) = \frac{1}{1 + e^{-z}}$$

En esta ecuación, z representa la combinación lineal de las variables independientes ponderadas por sus respectivos coeficientes:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k,$$

donde:

- $\beta_0, \beta_1, \beta_2, \ldots, \beta_k$, son los coeficientes de regresión que se deben estimar.
- $x_1, x_2, \ldots, x_k$, son los valores de las variables independientes.
- k es el número de variables independientes.

Buscamos el mejor valor posible para los parámetros $\beta_0, \beta_1, \beta_2, \ldots, \beta_k$ que definen el modelo logístico:

$$P(Y = 1|X_1, \ldots, X_k) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k)}}$$

$$P(Y = 0|X_1, \ldots, X_k) = 1 - P(Y = 1|X_1, \ldots, Xk)$$

La razón de probabilidades (odds-ratio) cuantifica el grado de asociación entre poseer o no la propiedad A y poseer o no la propiedad B. Para calcular odds-ratio se define como:

$$\frac{P(Y = 1|X_1, \ldots, X_k)}{P(Y = 0|X_1, \ldots, X_k)} = e^{(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k)}$$

Odds $\in [0,\infty]$:

- Odds < 1: Y $= 1$ es menor probable que Y $= 0$
- Odds $= 1$: Y $= 1$ es igual de probable que Y $= 0$
- Odds > 1: Y $= 1$ es más de probable que Y $= 0$

Es importante tener en cuenta que el odds ratio proporciona información sobre la relación entre la variable independiente y el evento de interés, pero no establece causalidad. Además, el cálculo del odds ratio se basa en la suposición de linealidad en el logit de la regresión logística.

Se define logit como el logaritmo del odds:

$$log\left(\frac{P(Y = 1|X_1, \ldots, X_k)}{P(Y = 0|X_1, \ldots, X_k)}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k$$

logit $\in$ (-∞, +∞.

El logit permite interpretar el signo de los coeficientes $\beta_0, \beta_1, \beta_2, \ldots, \beta_k$:

- $\beta_j > 0$ significa que cuanto más grande es el valor de X_j mayor es la probabilidad $P(Y = 1|X_1, \ldots, X_k)$

- $\beta_j < 0$ significa que cuanto más grande es el valor de X_j menor es la probabilidad $P(Y = 1|X_1, ..., X_k)$

5.2.1 Estimación

Los coeficientes $\beta_0, \beta_1, \beta_2, ..., \beta_k$ se obtienen a través del método de máxima verosimilitud:

$$\ell(\beta_0, \beta_1, \beta_2, ..., \beta_k) = \prod_{i:y_i=1} p(x_i) \prod_{i´:y_{i´}=0} (1 - p(x_{i´}))$$

siendo $p(x_i) = P(Y = 1|X_1 = X_{i1}, ..., X_k = x_{ik})$.

Una vez tenemos las estimaciones para $\beta_0, \beta_1, \beta_2, ..., \beta_k$, para nuevos valores de los predictores, la probabilidad de que la variable respuesta sea igual a 1 puede ser predicha sustituyendo los valores de los predictores en la ecuación.

La función logística transforma el valor z en un rango de 0 a 1, lo que se interpreta como la probabilidad de que la variable dependiente sea igual a 1. Si f(z) es mayor que el umbral, se clasifica como 1, y si f(z) es menor o igual al umbral, se clasifica como 0. Por defecto se suele elegir un umbral de 0.5, pero se puede utilizar el punto de corte entre la curva de especificidad y la curva de sensibilidad.

5.2.2 Residuos

Los residuos se refieren a las diferencias entre los valores observados y los valores predichos por el modelo. En lugar de utilizar la diferencia directa entre los valores observados y predichos, se utilizan los residuos de desviación.

En el método de regresión logística, los residuos se pueden calcular utilizando diferentes enfoques. Dos formas comunes de calcular los residuos son:

- Residuos de desviación:

Los residuos de desviación se calculan restando la respuesta observada (0 o 1 en regresión logística binaria) de la probabilidad ajustada por el modelo.

Sea y_k la respuesta observada (0 o 1) para la k-ésima observación y p_k la probabilidad ajustada por el modelo para esa misma observación.

El residuo de desviación (D_k) para la k-ésima observación se calcula como:

$$D_k = y_k - p_k$$

- Residuos estandarizados:

Los residuos estandarizados se calculan dividiendo los residuos de desviación por la raíz cuadrada de la varianza estimada de los residuos.

Sea S^2 la varianza estimada de los residuos, calculada de la siguiente manera:

$$S^2 = \frac{1}{n} \sum \frac{(y_k - p_k)^2}{p_k \cdot (1 - p_k)},$$

donde:

- n es el número de observaciones del conjunto de datos.
- y_k es la respuesta observada (0 o 1) para la k-ésima observación.
- p_k es la probabilidad ajustada por el modelo para la k-ésima observación.

El residuo estandarizado (S_k) para la k-ésima observación se calcula como:

$$S_k = \frac{D_k}{S^2}$$

Es importante destacar que los residuos en la regresión logística no se interpretan de la misma manera que en la regresión lineal. En lugar de medir la discrepancia absoluta entre los valores observados y predichos, los residuos en la regresión logística están relacionados con la deviance y la varianza de las predicciones. Estos residuos se utilizan principalmente para evaluar la calidad del ajuste del modelo y realizar diagnósticos adicionales.

5.3 Regresión Logística Multinomial

El modelo de regresión logística multinomial cuenta con los siguientes elementos:

- $Y_1, \dots, Y_n$ son vectores independientes tales que:

$$Y_i = \begin{pmatrix} Y_{i1} \\ \vdots \\ Y_{iq-1} \end{pmatrix} \stackrel{d}{=} M_q(m_i), \qquad \pi_i = \begin{pmatrix} \pi_{i1} \\ \vdots \\ \pi_{iq-1} \end{pmatrix}, i = 1, \dots, n$$

$m_1, \dots, m_n$ conocidos. Como la distribución de Y_i es multinomial, se verifica que $Y_{iq} = m_i - \sum_{j=1}^{q-1} Y_{ij}$ y $\pi_{iq} = 1 - \sum_{j=1}^{q-1} \pi_{ij}$, $i = 1, \dots, n$.

- Para cada categoría `j` hay un conjunto de `p` variables explicativas que se suponen conocidas.

$$(X_{11}, \dots, X_{1p}), \dots, (X_{j1}, \dots, X_{jp}), \dots, (X_{q-11}, \dots, X_{q-1p}),$$

de modo que sus puntuaciones en la unidad muestral i-ésima son

$$(x_{i11}, \dots, x_{i1p}), \dots, (x_{ij1}, \dots, x_{ijp}), \dots, (x_{iq-11}, \dots, x_{iq-1p}),$$

donde los índices corresponden a:

- $i = 1, \dots, n$ para las observaciones,
- $j = 1, \dots, q-1$ para las categorías,
- $k = 1, \dots, p$ para las variables auxiliares.

Los vectores de datos correspondientes a la observación i-ésima son:

$$y_i = \begin{pmatrix} y_{i1} \\ \vdots \\ y_{iq-1} \end{pmatrix}, x_{i1}^t \begin{pmatrix} 1 \\ x_{i11} \\ \vdots \\ x_{i1-p} \end{pmatrix}, \dots, x_{iq-1}^t \begin{pmatrix} 1 \\ x_{iq-11} \\ \vdots \\ x_{iq-1p} \end{pmatrix}.$$

- Existe un vector de parámetros $\beta_j = (\beta_{j0}, \beta_{j1}, \dots, \beta_{jp})´$ para cada categoría j que verifique:

$$log\frac{\pi_{ij}}{\pi_{iq}} = \beta_{j0} + \sum_{k=1}^{p} \beta_{jk} x_{ijk} = x_{ij}\beta_j, j = 1, \dots, q-1,$$

o equivalentemente

$$\pi_{ij} = \frac{exp\{x_{ij}\beta_j\}}{1 + \sum_{j=1}^{q-1} exp\{x_{ij}\beta_j\}}, j = 1, \dots, q-1, \pi_{iq} = \frac{1}{1 + \sum_{j=1}^{q-1} exp\{x_{ij}\beta_j\}}.$$

Si definimos

$$X_i = \begin{pmatrix} x_{i1} & & & \\ & x_{i2} & & \\ & & \ddots & \\ & & & x_{iq-1} \end{pmatrix}_{(q-1)\times(p+1)(q-1)}, \theta_i = \begin{pmatrix} \theta_{i1} \\ \vdots \\ \theta_{iq-1} \end{pmatrix}, \theta_{ij} = log\frac{\pi_{ij}}{\pi_{iq}}$$

y $\beta = (\beta_1^t, \dots, \beta_{q-1}^t)^t = (\beta_{1,0}, \beta_{1,1}, \dots, \beta_{1,p}, \dots, \beta_{q-1,0}, \beta_{q-1,1}, \dots, \beta_{q-1,p})^t$,
el modelo resultante es $\theta_i = X_i\beta$, $i =, \dots, n$.

5.3.1 Estimación

Para estimar los coeficientes β, se utiliza la técnica de máxima verosimilitud. El objetivo es encontrar los coeficientes que maximicen la función de verosimilitud logarítmica. La función de verosimilitud para la regresión logística multinomial es:

$$\ell(\beta; y_1, \dots, y_n) = \prod_{i=!}^{n} exp\left\{ \sum_{j=1}^{q-1} y_{ij} log\frac{\pi_{ij}(\beta)}{\pi_{iq}(\beta)} + m_i log\pi_{iq}(\beta) + log\frac{m_i!}{y_{i1}! \dots y_{iq}!} \right\}$$

5.3.2 Residuos

Sea $\widehat{\mu_{ij}} = m_i\widehat{\pi_{ij}}(\hat{\beta})$ el estimador de máxima verosimilitud de $E[Y_i]$ en el modelo, consideraremos los siguientes residuos:

- Residuos de Pearson

Los residuos Pearson son una medida de discrepancia entre las observaciones y las probabilidades predichas por el modelo en la regresión logística multinomial. Se calculan dividiendo la diferencia entre la observación y la probabilidad predicha por la raíz cuadrada de la varianza de la probabilidad predicha. La fórmula para los residuos Pearson es la siguiente:

$$r_{ij}^P = \frac{y_{ij} - \hat{\mu}_{ij}}{\sqrt{\hat{V}[Y_{ij}]}} = \frac{y_{ij} - m_i\pi_{ij}(\hat{\beta})}{\sqrt{m_i\pi_{ij}(\hat{\beta})(1 - \pi_{ij}(\hat{\beta}))}}, i = 1, \dots, n, j = 1, \dots, q-1.$$

- Residuos desviación

Los residuos desviación son una forma común de residuos utilizados en la regresión logística multinomial. Estos residuos se basan en la diferencia entre el logaritmo de las probabilidades predichas y el logaritmo de las probabilidades observadas. Se calculan de la siguiente manera:

$$r_{ij}^{D} = signo(y_{ij} - \hat{\mu}_{ij})\sqrt{D_i}, i = 1, \dots, n, j =, \dots, q-1.$$

donde

$$D_i = 2\left\{\sum_{j=1}^{q-1} y_{ij}\left(log\frac{y_{ij}}{y_{iq}} - \hat{\beta}_{jk}x_{ijk}\right) + m_i\left[log\frac{y_{iq}}{m_i} + log\left(1 + \sum_{j=1}^{q-1}\left\{\hat{\beta}_{j0} + \sum_{j=1}^{q-1}\hat{\beta}_{jk}x_{ijk}\right\}\right)\right]\right\}$$

5.4 Condiciones

Es importante que las observaciones sean independientes entre sí, es decir, que no estén influenciadas por las demás. Cada observación debe ser tomada de forma independiente para garantizar la validez de los resultados.

Es necesario que exista una relación lineal entre el logaritmo natural de odds y la variable continua que estamos estudiando. Patrones que siguen una forma en U claramente violan esta condición y pueden afectar la precisión de los resultados.

La regresión logística no requiere que la variable continua independiente siga una distribución normal.

5.5 Ventajas

1. Interpretabilidad: La regresión logística proporciona coeficientes que indican la influencia relativa de cada variable independiente en la probabilidad de ocurrencia del evento. Esto permite interpretar el impacto de cada variable en las predicciones.
2. Manejo de variables categóricas: La regresión logística puede manejar tanto variables numéricas como categóricas. Las variables categóricas se codifican como variables dummy, lo que facilita su inclusión en el modelo.
3. Eficiencia computacional: La regresión logística es computacionalmente eficiente, lo que significa que puede manejar conjuntos de datos grandes y es escalable para problemas con muchas variables independientes.
4. Buen funcionamiento con datos linealmente separables: Si los datos son linealmente separables, es decir, se pueden separar perfectamente utilizando una línea recta en un gráfico, la regresión logística puede proporcionar buenos resultados.

5.6 Desventajas

1. Suposiciones lineales: La regresión logística asume una relación lineal entre las variables independientes y el logit de la probabilidad del evento. Si la relación es no lineal, el rendimiento del modelo puede verse comprometido.
2. Sensibilidad a valores atípicos: La regresión logística puede ser sensible a valores atípicos en los datos. La presencia de valores extremos puede afectar la estimación de los coeficientes y, por lo tanto, las predicciones del modelo.

3. No adecuada para relaciones complejas: La regresión logística es un modelo relativamente simple y no puede capturar relaciones complejas entre las variables independientes y la variable dependiente. Si el problema tiene relaciones no lineales o interacciones complejas, es posible que la regresión logística no sea la mejor opción.

4. Requiere variables independientes relevantes: La regresión logística se basa en la suposición de que las variables independientes incluidas en el modelo son relevantes para predecir el evento. Si se incluyen variables irrelevantes, puede haber un impacto negativo en el rendimiento del modelo.

5.7 Programación en R

Empezaremos ajustando un modelo de regresión logística con los valores de la muestra.

```
#3
glm.fits <- glm(
  Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
  data = train_Weekly, family = binomial
)
step(glm.fits, direction="backward", k=2)
```

```
## Start:  AIC=1356.33
## Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume
##
##          Df Deviance    AIC
## - Lag3    1   1342.6 1354.6
## - Lag4    1   1343.5 1355.5
## - Lag5    1   1344.0 1356.0
## <none>        1342.3 1356.3
## - Lag2    1   1344.6 1356.6
## - Volume  1   1345.1 1357.1
## - Lag1    1   1346.9 1358.9
##
## Step:  AIC=1354.61
## Direction ~ Lag1 + Lag2 + Lag4 + Lag5 + Volume
##
##          Df Deviance    AIC
## - Lag4    1   1343.6 1353.6
## - Lag5    1   1344.3 1354.3
## <none>        1342.6 1354.6
## - Lag2    1   1345.1 1355.1
## - Volume  1   1345.2 1355.2
## - Lag1    1   1347.3 1357.3
##
## Step:  AIC=1353.64
## Direction ~ Lag1 + Lag2 + Lag5 + Volume
##
##          Df Deviance    AIC
## - Lag5    1   1345.1 1353.1
## <none>        1343.6 1353.6
## - Volume  1   1345.9 1353.9
## - Lag2    1   1346.0 1354.0
```

```
## - Lag1    1   1348.0 1356.0
##
## Step:  AIC=1353.14
## Direction ~ Lag1 + Lag2 + Volume
##
##          Df Deviance    AIC
## - Volume  1   1347.0 1353.0
## <none>        1345.1 1353.1
## - Lag2    1   1347.8 1353.8
## - Lag1    1   1349.4 1355.4
##
## Step:  AIC=1352.96
## Direction ~ Lag1 + Lag2
##
##        Df Deviance    AIC
## <none>      1347.0 1353.0
## - Lag2  1   1350.5 1354.5
## - Lag1  1   1350.5 1354.5
```

```
##
## Call:  glm(formula = Direction ~ Lag1 + Lag2, family = binomial, data = train_Weekly)
##
## Coefficients:
## (Intercept)         Lag1         Lag2
##     0.21109     -0.05421     0.05384
##
## Degrees of Freedom: 984 Total (i.e. Null);  982 Residual
## Null Deviance:       1355
## Residual Deviance: 1347  AIC: 1353
```

Se crea un modelo seleccionando todas las variables y comprobamos sus respectivos p-valores. Podemos ir eliminando variables auxiliares de una en una, pero es más sencillo usar un procedimiento automático. Así que se aplica la regresión por pasos "hacia atrás" basada en el AIC del cual nos quedaremos con aquel modelo que menor AIC tenga.

Con los resultados obtenidos (Cód.3) vemos que mediante el criterio del menor AIC nos quedamos con las variables `Lag1` y `Lag2`. Así que para los próximos métodos, nuestro modelo se reducirá a `Direction` $\sim$ `Lag1` + `Lag2`.

```
#4
glm.fits <- glm(
  Direction ~ Lag1 + Lag2,
  data = train_Weekly, family = binomial
)
summary(glm.fits)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2, family = binomial, data = train_Weekly)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -1.6149  -1.2565   0.9989   1.0875   1.5330
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.21109    0.06456   3.269  0.00108 **
## Lag1        -0.05421    0.02886  -1.878  0.06034 .
## Lag2         0.05384    0.02905   1.854  0.06379 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1354.7  on 984  degrees of freedom
## Residual deviance: 1347.0  on 982  degrees of freedom
## AIC: 1353
##
## Number of Fisher Scoring iterations: 4
```

Con el modelo ya propuesto utilizamos el código 4 para acceder a varios resultados y diagnósticos, como los coeficientes estimados, los valores ajustados, los residuos, las pruebas de significancia y los intervalos de confianza, entre otros. Estos resultados se pueden utilizar para interpretar el modelo y realizar inferencias sobre los efectos de las variables predictoras en la variable de respuesta.

En la salida de R (Cód.4) podemos ver que después de la reducción de variables realizada (Cód.3) las dos que han quedado son significativas y que el modelo resultante es el siguiente: $Y = 0.21109 - 0.05421Lag1 + 0.05384Lag2$.

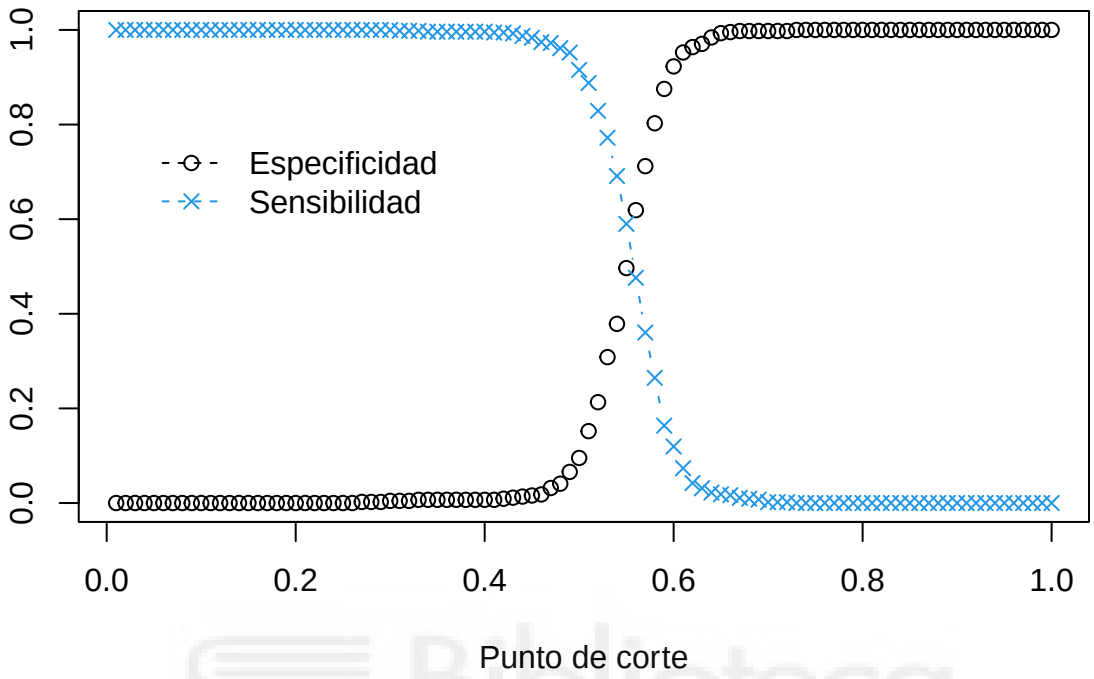
```r
#5
predictions <- predict(glm.fits, newdata = test_Weekly, type = "response")
```

Nos permite hacer predicciones o estimaciones basadas en el modelo estadístico ajustado previamente, permitiendo utilizar el modelo para predecir valores de respuesta para nuevas observaciones.

```r
#6
train_Weekly$Direction <- ifelse(train_Weekly$Direction == "Up", 1, 0)
tot1<-sum(train_Weekly$Direction)
tot0<-sum((1-train_Weekly$Direction))
n<-100
sensib<-rep(1,n)
especif<-rep(1,n)
corte<-rep(1,n)
for (i in 1:n) {
 corte[i]<-i/n
 sensib[i]<-sum(train_Weekly$Direction * (predict(glm.fits,type="response") > corte[i]))
 especif[i]<-sum((1-train_Weekly$Direction) * (predict(glm.fits,type="response") <= cort
}

plot(corte,especif, main = "", pch=1,col = 1, lty = 2,xlab = "Punto de corte", ylab = "
lines(corte,sensib, type = "b", col = 4, pch = 4, lty = 2)
text <- c("Especificidad", "Sensibilidad")
```

```
legend( 0.05, 0.8, text, col = c(1,4), pch=c(1,4), lty = c(2,2), bty="n")
```



Para poder calcular el umbral que nos permita clasificar los datos en 1 ó 0 (Cód.6) lo primero que hay que hacer es pasar los datos cualitativos de "Up" y "Down" en binarios. Después, se calcula la sensibilidad y la especificidad para cada corte. Graficamos los puntos, y el umbral que necesitamos será el punto de corte de ambas rectas.

Como podemos ver el punto de corte de ambas rectas se sitúa entre 0.5 y 0.6, pero en el gráfico es difícil saber cuál es el corte con exactitud, así que una forma de verlo más claramente es creando un dataframe de corte, sensibilidad y especificidad y comparando los valores de corte donde sensibilidad y especificidad sean iguales o parecidos.

```
#7
aux2<-data.frame(corte=corte, sensib=sensib, especif=especif)
Zonacorte<-subset(aux2,0.5<corte & corte<0.6); Zonacorte
```

```
##    corte    sensib   especif
## 51  0.51 0.8878676 0.1519274
## 52  0.52 0.8290441 0.2131519
## 53  0.53 0.7720588 0.3083900
## 54  0.54 0.6911765 0.3786848
## 55  0.55 0.5900735 0.4965986
## 56  0.56 0.4761029 0.6190476
## 57  0.57 0.3602941 0.7120181
## 58  0.58 0.2647059 0.8027211
## 59  0.59 0.1636029 0.8752834
```

En este caso, nos quedamos con 0.55 ya que la diferencia entre ambos es de 0.09.

```r
#8
library(caret)
umbral <- 0.55
predictedLabels <- ifelse(predictions > umbral, "Up", "Down")
actualLabels <- test_Weekly$Direction
confusionMatrix <- table(actualLabels, predictedLabels)
accuracy <- sum(diag(confusionMatrix)) / sum(confusionMatrix)
print(confusionMatrix)
```

```
##             predictedLabels
## actualLabels Down Up
##         Down   22 21
##         Up     25 36
```

```r
print(paste("Precisión:", accuracy))
```

```
## [1] "Precisión: 0.557692307692308"
```

Con el umbral calculado previamente (Cód.6), usaremos la librería `caret` para clasificar las estimaciones calculadas (Cód.5) y compararlas con los datos reales, creando una matriz de clasificación con los aciertos y fallos cometidos en la predicción.

El resultado es que el 55.77% de los movimientos semanales se han predicho correctamente.

6. Análisis Discriminante Lineal

6.1 Introducción

El análisis discriminante lineal (LDA, por sus siglas en inglés, Linear Discriminant Analysis) es una técnica de clasificación supervisada utilizada para encontrar una combinación lineal de variables que mejor separe dos o más grupos o clases diferentes.

6.1.1 Análisis Discriminante Lineal para p = 1

Supongamos que p = 1, es decir, solo tenemos un predictor. Necesitamos obtener la estimación de $f_k(x)$. Para ello, utilizaremos el Teorema de Bayes para sustituir dicha estimación y así poder obtener $p_k(x)$.

El teorema de Bayes establece que:

$$Pr(Y = k|X = x) = \frac{\pi_k f_k}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

Luego clasificaremos una observación en la clase para la cual $p_k(x)$ es mayor. Para estimar $f_k(x)$, primero haremos algunas suposiciones sobre su forma. Asumimos que $f_k(x)$ es normal o gaussiana. En el ajuste unidimensional, la densidad normal toma la forma:

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} exp\left(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right)$$

donde μ_k y σ_k^2 son los parámetros de la media y la varianza para la k-ésima clase. Además, asumimos que $\sigma_1^2 = \cdots = \sigma_k^2$, es decir, tienen varianza compartida en todas las `k` clases, por lo que para simplificar podemos denotarlo como σ^2. Remplazando la densidad normal en el Teorema de Bayes, encontramos que:

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{1}{2\sigma^2}(x-\mu_k)^2\right)}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{1}{2\sigma^2}(x-\mu_l)^2\right)}$$

π_k denota la probabilidad previa de que una observación pertenezca a la k-ésima clase, no confundir con la constante matemática, $\pi = 3.14....$

El clasificador de Bayes implica asignar una observación X = x a la clase para la cual $p_k(x)$ es mayor. Tomando el logaritmo de $p_k(x)$ y reorganizando los términos, demostramos que esto es equivalente a asignar la observación a la clase para la cual

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + log(\pi_k)$$

es mayor. Por ejemplo, si K = 2 y $\pi_1 = \pi_2$, entonces el clasificador de Bayes asigna una observación a la clase 1 si $2x(\mu_1 - \mu_2) > \mu_1^2 - \mu_2^2$, y a la clase 2 en caso contrario.

La frontera de decisión de Bayes es el punto para el cual $\delta_1(x) = \delta_2(x)$; se puede demostrar que esto equivale a:

$$x = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} = \frac{\mu_1 + \mu_2}{2}$$

En la práctica, incluso si estamos seguros de nuestra suposición de que X se extrae de una distribución gaussiana dentro de cada clase, para aplicar el clasificador de Bayes todavía tenemos que estimar los parámetros $\mu_1, ..., \mu_K$, $\pi_1, ..., \pi_K$ y σ^2. El método de análisis discriminante lineal se aproxima al clasificador de Bayes ajustando las estimaciones π_k, μ_k y σ^2 en $\delta_k(x)$. Utilizando las siguientes estimaciones:

$$\hat{\mu_k} = \frac{1}{n_k} \sum_{i:y_i=k} x_i$$

$$\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu_k})^2$$

donde `n` es el número total de observaciones de la estimación de los parámetros del modelo y n_k es el número de observaciones de la estimación de los parámetros en la k-ésima clase. La estimación de μ_k es simplemente el promedio de todas las observaciones de la k-ésima clase, mientras que $\hat{\sigma}^2$ es el promedio ponderado de las varianzas muestrales para cada una de K clases.

A veces conocemos las probabilidades de pertenencia a la clase $\pi_1, ..., \pi_K$, la cual podemos utilizar directamente. Pero, en ausencia de información adicional, el análisis discriminante lineal estima π_k utilizando la proporción de las observaciones que pertenecen a la k-ésima clase. Es decir, $\hat{\pi_k} = \frac{n_k}{n}$.

Con todo esto, definimos la regla de clasificación del análisis discriminante lineal como:

$$LDA(X) = \mathbb{I}(X - \frac{1}{2}(\hat{\mu}_1 + \hat{\mu}_2)^T \hat{\sum}^{-1} (\hat{\mu}_1 - \hat{\mu}_2) + log\frac{\hat{\pi}_1}{\hat{\pi}_2} > 0)$$

donde $\mathbb{I}(z > 0)$ es la función indicadora o característica, que toma el valor 1 si $z > 0$ y 0 en caso contrario.

6.1.2 Análisis Discriminante Lineal para p > 1

Ahora supongamos que el clasificador LDA tiene múltiples predictores. Para hacer esto, supondremos que $X = (X_1, X_2, ..., X_p)$ se extrae de una distribución gaussiana multivariante (o normal multivariante), con un vector medio específico de clase y una matriz de covarianza común.

La distribución gaussiana multivariante asume que cada predictor individual sigue una distribución unidimensional, como en $f_k(x)$, con cierta correlación entre cada par de predictores.

Para indicar que una variable aleatoria `X` de dimensión `p` tiene una distribución gaussiana multivariada, escribimos $X \sim N(\mu, \Sigma)$. En este caso $E(X) = \mu$ es la media de X (un vector con `p` componentes), y $Cov(X) = \Sigma$ es la matriz de covarianza `p x p` de `X`. Se define la densidad gaussiana multivariante como:

$$f(x) = \frac{1}{(2\pi)^{\frac{p}{2}} |\Sigma|^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu))$$

En el caso de p > 1 predictores, el clasificador LDA asume que las observaciones en la k-ésima clase se extraen de una distribución gaussiana multivariante $N(\mu_k, \Sigma)$, donde μ_k es un vector medio específico de clase y Σ es una matriz de covarianza que es común a todas las clases de K. Reemplazando la función de densidad para la k-ésima clase, $f_k(X = x)$, en el Teorema de Bayes, se revela que el clasificador de Bayes asigna una observación $X = x$ a la clase para la cual $\delta_k(x)$ es mayor.

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + log\pi_k$$

Siguiendo el mismo procedimiento que el mostrado con un solo predictor, se estiman los parámetros $\mu_1, ..., \mu_K$, $\pi_1, ..., \pi_K$ y Σ.

$$\hat{\pi_k} = \frac{n_k}{n}$$

$$\hat{\mu_k} = \frac{1}{n_k}\sum_{i:y_i=k} x_i$$

$$\hat{\sum} = \frac{1}{n-K}\sum_{k=1}^{K}\sum_{i:y_i=k}(x_i - \hat{\mu_k})(x_i - \hat{\mu_k})^T$$

6.4 Precisión

Para evaluar la precisión de un modelo de análisis discriminante lineal (LDA), generalmente se utilizan medidas de rendimiento comunes como la precisión, la matriz de confusión o clasificación y las métricas derivadas de ella (como la tasa de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos), y posiblemente curvas de precisión-recuperación (PR) o curvas características de operación del receptor (ROC).

Calculamos la precisión del modelo comparando las etiquetas de clase predichas con las etiquetas verdaderas del conjunto de prueba. La precisión se calcula como el número de predicciones correctas dividido por el número total de ejemplos en el conjunto de prueba.

Calculamos la matriz de clasificación, que muestra el número de ejemplos que fueron clasificados correctamente e incorrectamente para cada clase. Esto brinda una visión más detallada del rendimiento del modelo.

A partir de la matriz de clasificación, podemos calcular métricas como la tasa de verdaderos positivos (sensibilidad), tasa de falsos positivos, tasa de verdaderos negativos y tasa de falsos negativos. Estas métricas proporcionan información adicional sobre el rendimiento del modelo.

Si tenemos datos con etiquetas de clase desequilibradas, también podemos trazar curvas PR y ROC para evaluar el rendimiento del modelo en diferentes umbrales de clasificación.

6.5 Condiciones

Para que un Análisis Discriminante Lineal (LDA) sea válido, es necesario que se cumplan ciertas condiciones:

- Cada variable predictora que se utiliza en el modelo debe seguir una distribución normal en cada una de las clases de la variable respuesta. En el caso de tener múltiples variables predictoras, las observaciones deben seguir una distribución normal multivariante en todas las clases.
- La varianza de las variables predictoras debe ser la misma en todas las clases de la variable respuesta. Si se utilizan múltiples variables predictoras, la matriz de covarianza debe ser igual en todas las clases.

Cuando no se cumple la condición de normalidad, la precisión del LDA se ve afectada, pero aun así puede lograr clasificaciones relativamente buenas.

6.6 Ventajas

1. Reducción de dimensionalidad: El LDA permite reducir la dimensionalidad de los datos al seleccionar los componentes principales más relevantes. Esto puede ser beneficioso cuando se trabaja con conjuntos de datos de alta dimensionalidad, ya que ayuda a eliminar características redundantes y mejorar la eficiencia computacional.
2. Preservación de la información discriminante: El LDA busca maximizar la separabilidad entre las clases al proyectar los datos en un nuevo espacio. Esto significa que intenta mantener la información que es relevante para la clasificación y descartar la información redundante o no discriminante.
3. Clasificación probabilística: El LDA estima las probabilidades a posteriori de pertenencia a cada clase. Esto proporciona una medida de confianza o incertidumbre en las predicciones, lo que puede ser útil en aplicaciones donde se requiere una evaluación más completa de las mismas.
4. Utilidad con muestras pequeñas: El LDA funciona bien incluso cuando el número de muestras de entrenamiento es pequeño en comparación con la dimensionalidad de los datos. Esto puede ser una ventaja en escenarios donde la recopilación de datos etiquetados es costosa o limitada.

6.7 Desventajas

1. Asunciones de linealidad y normalidad: El LDA asume que los datos se distribuyen normalmente y que las clases tienen matrices de covarianza iguales. Estas suposiciones pueden no cumplirse en algunos conjuntos de datos reales, lo que puede afectar la precisión del modelo.
2. Sensibilidad a valores atípicos: El LDA es sensible a los valores atípicos en los datos. Los valores atípicos pueden afectar la estimación de las matrices de covarianza y, en consecuencia, la separabilidad de las clases.
3. Limitaciones en problemas de clasificación no lineal: El LDA es un clasificador lineal y puede tener dificultades para capturar relaciones no lineales entre las características y las clases. En problemas donde la frontera de decisión es compleja o no lineal, el LDA puede no ser el enfoque más adecuado.
4. Requisito de clases bien separadas: El rendimiento del LDA puede verse afectado si las clases se superponen considerablemente en el espacio de características. Si las clases son altamente superpuestas, el LDA puede tener dificultades para encontrar una buena separación entre ellas.

6.8 Programación en R

Ajustamos el modelo LDA (Cód.9) en los datos de `Weekly` usando solo las observaciones antes de 2009. Además de cargar la librería `MASS` para poder utilizar la función `lda()`

```
#9
library(MASS)
lda.fit <- lda(Direction ~ Lag1 + Lag2, data = train_Weekly)
lda.fit
```

```
## Call:
## lda(Direction ~ Lag1 + Lag2, data = train_Weekly)
##
## Prior probabilities of groups:
##      Down        Up
## 0.4477157 0.5522843
##
## Group means:
##               Lag1        Lag2
## Down  0.289444444 -0.03568254
## Up   -0.009213235  0.26036581
##
## Coefficients of linear discriminants:
##             LD1
## Lag1 -0.3013148
## Lag2  0.2982579
```

La salida del LDA (Cód.9) indica que $\pi_1 = 0.448$ y $\pi_2 = 0.552$; es decir, el 44.8% de las observaciones de la muestra corresponden a semanas en los que el mercado bajó. Estas estimaciones sugieren que existe una tendencia a que cuando el mercado aumenta, el rendimiento de la semana anterior será negativo y la anterior a esa, positivo. Justo lo contrario pasa cuando el mercado cae; el rendimiento de la semana anterior será positivo y la anterior a esa, negativo. Los coeficientes del discriminante lineal nos demuestran que si $-0.301 \times Lag1 + 0.298 \times Lag2$ es grande, entonces el clasificador LDA predecirá un incremento del mercado, y si es pequeño, entonces predecirá una bajada.

Una vez visto esto, realizamos la predicción con los parámetros obtenidos.

```
#10
lda.pred <- predict(lda.fit, test_Weekly)
lda.class <- lda.pred$class

confusionMatrix(lda.class, test_Weekly$Direction)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Down Up
##       Down    7  8
##       Up     36 53
##
##                Accuracy : 0.5769
##                  95% CI : (0.4761, 0.6732)
```

```
##     No Information Rate : 0.5865
##     P-Value [Acc > NIR] : 0.6193
##
##                   Kappa : 0.035
##
##  Mcnemar's Test P-Value : 4.693e-05
##
##             Sensitivity : 0.16279
##             Specificity : 0.86885
##          Pos Pred Value : 0.46667
##          Neg Pred Value : 0.59551
##              Prevalence : 0.41346
##          Detection Rate : 0.06731
##    Detection Prevalence : 0.14423
##       Balanced Accuracy : 0.51582
##
##        'Positive' Class : Down
##
```

```
table(lda.class, test_Weekly$Direction)
```

```
##
## lda.class Down Up
##      Down    7  8
##      Up     36 53
```

```
mean(lda.class == test_Weekly$Direction)
```

```
## [1] 0.5769231
```

Para obtener las predicciones se utilizan los coeficientes del modelo ajustado a los datos de muestra y se utilizan en los datos de prueba.

Como podemos observar (Cód.10), las predicciones del Análisis Discriminante Lineal nos dan un porcentaje de acierto del 57.69%.

7. Análisis Discriminante Cuadrático

7.1 Introducción

El análisis discriminante cuadrático (QDA, por sus siglas en inglés, Quadratic Discriminant Analysis) es una técnica estadística de clasificación utilizada para clasificar observaciones en diferentes grupos o categorías. Es una extensión del análisis discriminante lineal, que asume que las variables predictoras se distribuyen normalmente y que las matrices de covarianza de las variables son iguales en todos los grupos.

En el análisis discriminante cuadrático, se parte de la premisa de que las variables predictoras están relacionadas de manera cuadrática con la pertenencia a un grupo específico. Esto significa que se modela la estructura cuadrática de los datos y se utilizan matrices de covarianza distintas para cada grupo.

Al igual que el análisis discriminante lineal, el clasificador QDA resulta de asumir que las observaciones de cada clase se extraen de una distribución gaussiana y de conectar las estimaciones de los parámetros en el teorema de Bayes para realizar la predicción. Por lo cual tenemos la siguiente función de densidad normal:

$$f(x) = \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma|^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

Sin embargo, a diferencia del clasificador LDA, QDA asume que cada clase tiene su propia matriz de covarianza. Es decir, asume que una observación de la k-ésima clase es de la forma $X \sim N(\mu_k, \Sigma_k)$, donde Σ_k es la matriz de covarianza para la k-ésima clase. Bajo este supuesto, el clasificador de Bayes asigna una observación $X = x$ a la clase para la cual $\delta_k(x)$ es mayor.

$$\delta_k(x) = -\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) - \frac{1}{2}log|\Sigma_k| + log\pi_k$$

$$= -\frac{1}{2}x^T\Sigma_k^{-1}x + x^T\Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k - \frac{1}{2}log|\Sigma_k| + log\pi_k$$

Con todo esto, definimos la regla del análisis discriminante cuadrático como:

$$QDA(X) = \mathbb{I}\left((X-\mu_1)^T\Sigma_1^{-1}(X-\mu_1) - (X-\mu_2)^T\Sigma_2^{-1}(X-\mu_2) + log\frac{|\Sigma_1|}{|\Sigma_2|} > 0\right)$$

donde $\mathbb{I}(z > 0)$ es la función indicadora o característica, que toma el valor 1 si $z > 0$ y 0 en caso contrario.

7.2 Precisión

Para evaluar la precisión del modelo QDA, podemos utilizar diversas métricas de evaluación comunes en el campo de la clasificación:

- Exactitud: Es la métrica más común y sencilla de entender. La exactitud se calcula dividiendo el número de predicciones correctas entre el número total de predicciones. Exactitud = (Predicciones correctas) / (Total de predicciones)

- Matriz de clasificación: Matriz de clasificación es una tabla que muestra la cantidad de verdaderos positivos (VP), verdaderos negativos (VN), falsos positivos (FP) y falsos negativos (FN). Estos valores se utilizan para calcular otras métricas como precisión, exhaustividad y puntuación F1.
- Precisión: La precisión mide la proporción de predicciones positivas que fueron correctas. Se calcula dividiendo los verdaderos positivos entre la suma de verdaderos positivos y falsos positivos. Precisión = TP / (TP + FP)
- Sensibilidad: La sensibilidad mide la proporción de ejemplos positivos que se clasificaron correctamente. Se calcula dividiendo los verdaderos positivos entre la suma de verdaderos positivos y falsos negativos. Sensibilidad = VP / (VP + FN)
- F1 Score: La puntuación F1 es una métrica que combina la precisión y la sensibilidad en un solo valor. Es útil cuando quieres tener en cuenta tanto los falsos positivos como los falsos negativos. Se calcula utilizando la siguiente fórmula: F1 Score = 2 * (Precisión * Sensibilidad) / (Precisión + Sensibilidad)

7.3 Condiciones y consideraciones

Para utilizar el clasificador QDA se debe tener en cuenta que los datos deben estar distribuidos en diferentes clases, y se espera que cada clase tenga una matriz de covarianza diferente. Además, es recomendable que los datos sean numéricos y continuos.

Las variables predictoras deben ser independientes entre sí dentro de cada clase, asumiendo que siguen una distribución normal multivariante dentro de cada clase.

Será necesario contar con un tamaño de muestra adecuado en cada clase para obtener resultados confiables.

7.4 Ventajas

1. Considera la covarianza de las variables: A diferencia del análisis discriminante lineal, el QDA tiene en cuenta la estructura de covarianza de las variables predictoras, lo que puede ser útil cuando las variables están correlacionadas entre sí.
2. Clasificación flexible: El QDA permite clasificar observaciones en múltiples grupos, lo que lo hace adecuado para problemas de clasificación con más de dos clases.
3. Toma en cuenta la distribución de los datos: El QDA asume que las variables predictoras siguen una distribución normal multivariada en cada clase, lo cual puede ser beneficioso cuando los datos cumplen con esta suposición.
4. Puede manejar muestras pequeñas: A diferencia de algunos otros algoritmos de aprendizaje automático, el QDA puede funcionar bien incluso cuando se tiene un número limitado de observaciones de entrenamiento.

7.5 Desventajas

1. Sensible a suposiciones de distribución: El QDA asume que las variables predictoras siguen una distribución normal multivariada en cada clase. Al igual que si las condiciones se cumplen es una ventaja. Si esta suposición no se cumple, las predicciones del modelo pueden ser inexactas.

2. Sensible a problemas de multicolinealidad: Si las variables predictoras están altamente correlacionadas entre sí, el QDA puede tener dificultades para estimar las matrices de covarianza, lo que puede afectar la precisión del modelo.
3. Puede sufrir de sobreajuste: Si el número de variables predictoras es mucho mayor que el número de observaciones de entrenamiento, el QDA puede sufrir de sobreajuste, lo que significa que el modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien a nuevos datos.
4. Sensibilidad a la presencia de valores atípicos: Los valores atípicos pueden afectar significativamente la estimación de las matrices de covarianza y las medias de las clases, lo que a su vez puede afectar la precisión y el rendimiento del modelo.

7.6 Programación en R

Ajustaremos un modelo QDA a los datos de `Weekly`. El QDA está implementado usando la función `qda()`, que también forma parte de la librería `MASS`. La sintaxis es idéntica a la de `lda()`

```
#11
library(MASS)
qda.fit <- qda(Direction ~ Lag1 + Lag2, data = train_Weekly)
qda.fit
```

```
## Call:
## qda(Direction ~ Lag1 + Lag2, data = train_Weekly)
##
## Prior probabilities of groups:
##      Down        Up
## 0.4477157 0.5522843
##
## Group means:
##               Lag1        Lag2
## Down  0.289444444 -0.03568254
## Up   -0.009213235  0.26036581
```

La salida (Cód.11) contiene las medias de los grupos. Pero no contiene los coeficientes de los discriminantes lineales, porque el clasificador QDA implica una función cuadrática de los predictores. La función `predict()` funciona exactamente de la misma forma que para el LDA.

```
#12
qda.class <- predict(qda.fit, test_Weekly)$class
table(qda.class, test_Weekly$Direction)
```

```
##
## qda.class Down Up
##      Down    7 10
##      Up     36 51
```

```
mean(qda.class == test_Weekly$Direction)
```

```
## [1] 0.5576923
```

Tenemos un porcentaje de acierto del 55.77% (Cód.10), peor que los vistos anteriormente.

8. Naive Bayes

8.1 Introducción

El clasificador de Naive Bayes es un método de clasificación de aprendizaje supervisado. Se basa en el teorema de Bayes y hace suposiciones ingenuas (naive) sobre la independencia condicional de las características.

Para entender los fundamentos matemáticos del clasificador de Naive Bayes, consideremos un problema de clasificación binaria en el que tenemos un conjunto de características (variables) independientes $X_1, X_2, ..., X_n$ y una variable de clase C con dos posibles valores, por ejemplo, "positivo" y "negativo".

El objetivo es calcular la probabilidad de que una instancia con características dadas pertenezca a cada una de las clases y luego asignar a la clase con la probabilidad más alta.

El clasificador de Naive Bayes utiliza el teorema de Bayes para calcular estas probabilidades. Según el teorema de Bayes:

$$P(C|X_1, X_2, ..., X_n) = \frac{P(C) \cdot P(X_1, X_2, ..., X_n|C)}{P(X_1, X_2, ..., X_n)}$$

Donde:

- $P(C|X_1, X_2, ..., X_n)$ es la probabilidad de que la instancia pertenezca a la clase C dado los atributos observados.
- $P(C)$ es la probabilidad a priori de la clase C.
- $P(X_1, X_2, ..., X_n|C)$ es la probabilidad de observar los atributos $x_1, x_2, ..., x_n$ dados que la instancia pertenezca a la clase C.
- $P(X_1, X_2, ..., X_n)$ es la probabilidad de observar los atributos $x_1, x_2, ..., x_n$ en general.

En el clasificador de Naive Bayes, se hace una suposición ingenua de que las características son independientes entre sí dada la clase. Esta suposición simplifica el cálculo de la probabilidad condicional $P(X_1, X_2, ..., X_n|C)$ utilizando la regla de la cadena:

$$P(X_1, X_2, ..., X_n|C) = P(X_1|C) \cdot P(X_2|C, X_1) \cdot ... \cdot P(X_n|C, X_1, X_2, ..., X_{n-1})$$

Usando la suposición de independencia condicional, podemos simplificar esta expresión a:

$$P(X_1, X_2, ..., X_n|C) = P(X_1|C) \cdot P(X_2|C) \cdot ... \cdot P(X_n|C)$$

Ahora, el clasificador de Naive Bayes asume que las probabilidades $P(X_i|C)$ se pueden estimar a partir de los datos estimados de los parámetros del modelo. Dependiendo del tipo de datos, se utilizan diferentes distribuciones de probabilidad para modelar estas probabilidades condicionales, como la distribución normal (Gaussiana) para características continuas

$$P(X_i = x|C) = \frac{1}{\sqrt{2\pi\sigma_{C,i}^2}} exp\left(-\frac{1}{2\sigma_{C,i}^2}(x - \mu_{C,i})^2\right)$$

donde $\mu_{C,i}$ es la media de la variable X_i para la clase C y $\sigma^2_{C,i}$ es la varianza de la variable X_i para la clase C, o la distribución multinomial para característica discretas

$$P(X_i = x|C) = \frac{n_{C,i}(x)}{\sum_{x'} n_{C,i}(x')}$$

donde $n_{C,i}(x)$ es el número de veces que el valor x de la variable X_i aparece en la clase C.

Finalmente, para tomar una decisión de clasificación, se compara la probabilidad posterior $P(C|X_1, X_2, ..., X_n)$ para cada clase posible y se asigna la clase con la probabilidad más alta.

8.2 Tipos de Clasificador de Naive Bayes

8.2.1 Naive Bayes Gaussiano

El Naive Bayes Gaussiano es una variante del clasificador de Naive Bayes que se utiliza cuando los atributos se distribuyen normalmente (o aproximadamente) dentro de cada clase.

El proceso de entrenamiento del Naive Bayes Gaussiano implica calcular los parámetros de la distribución normal para cada atributo en cada clase. Esto se realiza mediante la estimación de la media y la varianza de los valores de cada atributo dentro de cada clase.

Dado un conjunto de datos de muestra, se calculan los siguientes parámetros de la distribución normal para cada atributo en cada clase:

- Media (μ): Es el promedio de los valores del atributo en la clase C. Se calcula sumando todos los valores del atributo en la clase C y dividiéndolo por el número total de instancias en la clase C.
- Varianza (σ^2): Es una medida de la dispersión de los valores del atributo en la clase C. Se calcula sumando la diferencia al cuadrado entre cada valor del atributo y la media, y dividiéndolo por el número total de instancias en la clase C.

Es importante tener en cuenta que el Naive Bayes Gaussiano asume que los atributos se distribuyen normalmente dentro de cada clase y que la suposición "naive" de independencia condicional se mantiene entre los atributos dado la clase.

8.3 Condiciones

Para realizar un clasificador Naive Bayes, se deben tener en cuenta algunas condiciones y consideraciones:

- Independencia de las características: El algoritmo Naive Bayes asume que las características son independientes entre sí dado el valor de la variable de clase. Esta suposición puede ser problemática si las características están altamente correlacionadas.
- Datos categóricos o discretos: El clasificador Naive Bayes es más adecuado para datos categóricos o discretos. Aunque se puede aplicar a datos numéricos, generalmente se requiere una discretización previa de las variables continuas.
- Conjunto de datos de entrenamiento representativo: Es importante tener un conjunto de datos de entrenamiento representativo que refleje la distribución real de las clases en el dominio del problema. Un conjunto de datos desequilibrado puede afectar negativamente el rendimiento del clasificador.

- Manejo de valores faltantes: Naive Bayes asume que no hay valores faltantes en los datos de entrenamiento. Por lo tanto, es necesario manejar adecuadamente los valores faltantes antes de aplicar el algoritmo. Esto puede implicar la eliminación de instancias con valores faltantes o el uso de técnicas de imputación.
- Tamaño del conjunto de datos: Aunque Naive Bayes puede funcionar bien con conjuntos de datos pequeños, generalmente se desempeña mejor cuando se tienen grandes cantidades de datos de entrenamiento. Esto se debe a que el clasificador estima las probabilidades de las características y la variable de clase a partir de los datos de entrenamiento.
- Distribución de las características: Naive Bayes asume una distribución de características específica, como la distribución de Bernoulli para características binarias o la distribución multinomial para características discretas con múltiples categorías. Es importante verificar si estas distribuciones son apropiadas para los datos en cuestión.
- Supuestos de independencia: Naive Bayes supone que las características son independientes entre sí dada la variable de clase. Este supuesto puede no ser válido en muchos casos del mundo real. Si hay una fuerte dependencia entre las características, el rendimiento del clasificador puede verse afectado.
- Balance de clases: Naive Bayes no tiene en cuenta el desequilibrio en la distribución de clases. Si el conjunto de datos presenta un desequilibrio significativo entre las clases, puede ser necesario tomar medidas adicionales, como el muestreo estratificado, para abordar este problema.

8.4 Ventajas

1. Eficiencia computacional: El clasificador de Naive Bayes es rápido y eficiente en términos computacionales. El cálculo de las probabilidades condicionales es relativamente simple y rápido, lo que hace que sea adecuado para conjuntos de datos grandes.
2. Buen rendimiento con conjuntos de datos pequeños: El clasificador de Naive Bayes puede funcionar bien incluso con un número limitado de instancias de entrenamiento. Esto lo hace especialmente útil cuando se tiene un conjunto de datos pequeño.
3. Manejo de características irrelevantes: El clasificador de Naive Bayes asume que las características son independientes entre sí, lo que significa que puede manejar características irrelevantes o redundantes sin afectar significativamente su rendimiento. Esto hace que sea robusto en presencia de ruido o características irrelevantes.
4. Interpretabilidad: Naive Bayes proporciona una interpretación intuitiva y fácil de entender de las predicciones. Las probabilidades condicionales calculadas pueden ser interpretadas como la fuerza de la evidencia para cada clase.

8.5 Desventajas

1. Supuesto de independencia de características: La principal desventaja del clasificador de Naive Bayes es su supuesto de independencia entre las características. En muchos casos, esta suposición no se cumple en la práctica, lo que puede conducir a una precisión reducida de las predicciones.
2. Sensibilidad a datos de entrenamiento insuficientes: Aunque Naive Bayes funciona bien con conjuntos de datos pequeños, puede tener dificultades cuando hay una falta de datos

de entrenamiento representativos para algunas clases o combinaciones de características. Esto puede llevar a estimaciones inexactas de las probabilidades condicionales.

3. Limitación en la representación de relaciones complejas: Debido a su supuesto de independencia de características, Naive Bayes puede tener dificultades para capturar relaciones complejas o dependencias entre las características. Esto puede llevar a una precisión reducida en problemas donde las relaciones entre las características son importantes.

4. Sensibilidad a valores atípicos: El clasificador de Naive Bayes asume una distribución de características normal, lo que significa que puede verse afectado negativamente por valores atípicos o datos que no sigan una distribución normal.

8.6 Programación en R

Ajustamos un modelo Naive Bayes a los datos de `Weekly`. Naive Bayes se implementa mediante la función `naiveBayes()`, que es parte de la librería `e1071`. La sintaxis es idéntica a la de `lda()` y `qda()`. De forma predeterminada, esta implementación del clasificador de Naive Bayes modela cada característica cuantitativa mediante una distribución gaussiana. Sin embargo, también se puede usar un método de densidad de kernel para estimar las distribuciones.

```
#13
library(e1071)
nb.fit <- naiveBayes(Direction ~ Lag1 + Lag2, data = train_Weekly)
nb.fit
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##      Down        Up
## 0.4477157 0.5522843
##
## Conditional probabilities:
##       Lag1
## Y              [,1]     [,2]
##   Down  0.289444444 2.211721
##   Up   -0.009213235 2.308387
##
##       Lag2
## Y             [,1]     [,2]
##   Down -0.03568254 2.199504
##   Up    0.26036581 2.317485
```

La salida (Cód.13) contiene la media estimada y la desviación típica por cada variable en cada clase. Por ejemplo, la media para `Lag1` es 0.2894 para `Direction=Down`, y la desviación típica es 2.21.

```
#14
nb.class <- predict(nb.fit, test_Weekly)
```

```
table(nb.class, test_Weekly$Direction)
```

```
##
## nb.class Down Up
##     Down    3  8
##     Up     40 53
```

```
mean(nb.class == test_Weekly$Direction)
```

```
## [1] 0.5384615
```

Tras realizar validación cruzada (Cód.14), Naive Bayes no funciona muy bien con estos datos, con predicciones precisas el 53.85% de las veces.

9. K-Nearest Neighbors (K-Vecinos Más Cercanos)

9.1 Introducción

El método K-Nearest Neighbors (KNN) traducido como el método de los K vecinos más cercanos es un método de clasificación supervisada.

Cuando se tiene un individuo que se desea clasificar, se localiza en los datos de muestra (donde se conocen las clasificaciones) los K individuos más cercanos al individuo a clasificar. Por último, el individuo se clasifica en la categoría que es más común entre los K sujetos más cercanos.

Dado un número entero positivo K, y un conjunto de observaciones $x = (x_1, ..., x_k)$, el método KNN funciona de la siguiente manera:

Se identifican los K puntos del conjunto de muestra más cercanos a x, y los representamos por N_x. Estimamos la probabilidad condicionada para la clase j como la proporción de puntos en N_x cuyo valor es igual a j:

$$P(Y = j|X = x) = \frac{1}{K} \sum_{i \in N_x} \mathbb{I}(y_i = j),$$

siendo $\mathbb{I}(y_i = j) = 1$ si $y_i = j$ y 0 en otro caso. Por último, siguiendo la regla del clasificador de Bayes, KNN clasifica la observación x a la clase con la probabilidad más alta.

9.1.1 Selección de k

La selección del valor adecuado para el hiperparámetro k en el clasificador K-Nearest Neighbors (KNN) es una parte importante del proceso.

Se prueban diferentes valores de k y se evalúa su impacto en el rendimiento del modelo. A veces, un valor bajo de k puede llevar a un sobreajuste, mientras que un valor alto puede resultar en un sesgo. Se observa cómo se comporta el modelo en función de diferentes valores de k y se elige aquel que proporcione un equilibrio adecuado entre sesgo y varianza.

Para evaluar el impacto en el rendimiento del modelo utilizamos algunas estrategias como:

- Validación cruzada: La validación cruzada es una técnica que te permite evaluar el rendimiento del modelo utilizando diferentes divisiones de los datos de muestra y prueba. Se puede realizar una validación cruzada con diferentes valores de k y seleccionar aquel que ofrezca el mejor rendimiento promedio en términos de métricas de evaluación, como la precisión o el F1-score.
- Curva de validación: Se puede graficar el rendimiento del modelo en función de diferentes valores de k. Esto se conoce como curva de validación. Se observa cómo varían las métricas de evaluación a medida que se cambia el valor de k. Se busca el punto en el que el rendimiento se estabilice o alcance su punto máximo. Ese puede ser un buen valor para k.
- Regla de la raíz cuadrada: Una regla empírica comúnmente utilizada sugiere que el valor de k debe ser aproximadamente la raíz cuadrada del número total de muestras. Sin embargo, esta regla puede variar según el problema y los datos específicos, por lo que es importante utilizarla solo como una guía inicial y ajustar el valor de k según sea necesario.

9.1.2 Calculo de la distancia

La distancia en el clasificador K-Nearest Neighbors (KNN) es una medida utilizada para calcular la similitud entre objetos y determinar los vecinos más cercanos a una muestra de prueba. La distancia es fundamental en KNN ya que se utiliza para tomar decisiones de clasificación basadas en la cercanía de las muestras.

El propósito principal de utilizar la distancia en KNN es encontrar objetos similares en función de sus características. Cuanto más cercanas sean las características entre dos objetos, menor será la distancia entre ellos. Al calcular la distancia entre una muestra de prueba y los parámetros del modelo conocidos, el algoritmo KNN busca aquellos objetos que sean más similares o cercanos a la muestra de prueba.

La distancia más comúnmente utilizada en KNN es la distancia euclídea, que calcula la longitud del segmento que une dos puntos en un espacio n-dimensional.

$$D(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

Sin embargo, también se pueden utilizar otras medidas de distancia, como la distancia de Manhattan: $D(x, y) = (\sum_{i=1}^{m} |x_i - y_i|)$, la distancia de Minkowski: $D(x, y) = (\sum_{i=1}^{n} |x_i - y_i|)^{\frac{1}{p}}$ o la distancia de Hamming: $D_H(x, y) = (\sum_{i=1}^{k} |x_i - y_i|)$, dependiendo de las características y el tipo de datos que se estén tratando.

9.2 Condiciones

En el método de clasificación KNN es importante comprender las condiciones que deben cumplirse para aplicarlo correctamente. Estas condiciones están relacionadas con los datos de entrada, las características y la preparación previa necesaria para garantizar resultados precisos y confiables.

En dichas condiciones se requiere un conjunto de datos etiquetados, donde cada instancia tiene una etiqueta o clase asociada. Esto significa que se debe tener información sobre las clases a las que pertenecen los datos para poder evaluar y clasificar el modelo correctamente.

KNN funciona mejor con variables numéricas, ya que se basa en la distancia entre los puntos de datos para realizar la clasificación. Si los datos contienen características categóricas, se pueden convertir en variables numéricas. Además, es recomendable escalar las variables numéricas antes de aplicar KNN. El escalado es importante porque si las variables tienen diferentes rangos o magnitudes, algunas variables pueden dominar la medida de distancia y afectar los resultados del algoritmo. Se puede utilizar técnicas como la estandarización o la normalización para escalar los datos.

9.3 Ventajas

1. Simplicidad: K-Nearest Neighbors es un algoritmo relativamente simple de entender e implementar. No requiere una fase de entrenamiento costosa computacionalmente, ya que simplemente almacena los datos de entrenamiento en memoria.

2. No paramétrico: A diferencia de otros algoritmos, como la regresión lineal o los modelos basados en árboles de decisión, K-Nearest Neighbors no hace suposiciones explícitas so-

bre la distribución de los datos. Esto le permite funcionar bien en una amplia gama de problemas y adaptarse a diferentes tipos de datos.

3. Flexibilidad en la clasificación: K-Nearest Neighbors puede manejar tanto problemas de clasificación binaria como multiclase. También es capaz de manejar problemas con datos desbalanceados, donde hay una cantidad significativamente diferente de ejemplos en cada clase.

4. Capacidad para capturar estructuras y fronteras de decisión no lineales en los datos: lo que lo hace adecuado para problemas complejos donde las relaciones no pueden ser modeladas de manera lineal.

9.4 Desventajas

1. Sensible a la escala de los datos: K-Nearest Neighbors considera la distancia entre los puntos de datos para realizar predicciones. Si las características tienen diferentes escalas, aquellas con valores más grandes pueden tener un impacto desproporcionado en el cálculo de las distancias. Por lo tanto, es importante normalizar las características antes de aplicar K-Nearest Neighbors.

2. Sensible a la dimensionalidad: Con un aumento en el número de dimensiones (características) en los datos, la densidad de los puntos de datos se diluye. Esto puede hacer que el rendimiento de K-Nearest Neighbors disminuya, ya que la noción de vecindad se vuelve menos significativa.

3. Costoso en términos computacionales: Para realizar predicciones, K-Nearest Neighbors necesita buscar los K vecinos más cercanos en el conjunto de entrenamiento para cada punto de prueba. Esto puede ser computacionalmente costoso, especialmente cuando el conjunto de entrenamiento es grande y/o las dimensiones son altas.

4. Falta de interpretabilidad: K-Nearest Neighbors no proporciona una forma directa de interpretar las relaciones entre las características y las predicciones. Simplemente utiliza las etiquetas de los vecinos más cercanos sin generar un modelo explícito que capture las relaciones subyacentes.

9.5 Programación en R

Realizaremos KNN usando la función `knn()`, que es parte de la librería `class`. Esta función funciona de manera bastante diferente a las otras funciones de ajuste de modelo que hemos visto hasta ahora. En lugar de un planteamiento de dos pasos en el que primero ajustamos el modelo y luego usamos el modelo para hacer predicciones, `knn()` proporciona las predicciones mediante un solo comando. La función requiere cuatro entradas:

1. Una matriz que contiene los predictores asociados con los datos de muestra, etiquetado como `train.X`.

2. Una matriz que contiene los predictores asociados con los datos para los que deseamos hacer predicciones, etiquetado como `test.X`.

3. Un vector que contiene las etiquetas de clase para las observaciones de muestra, etiquetado como `train.Direction`.

4. Un valor para K, el número de vecinos más cercanos que utilizará el clasificador.

Para no estar probando uno a uno los distintos k para encontrar la mejor predicción, se programará un código que lo haga automáticamente.

```
#15
library(class)
set.seed(1)
X <- Weekly[2:3]
Y <- Weekly$Direction

train_indices <- Weekly$Year >= 1990 & Weekly$Year <= 2008
test_indices <- Weekly$Year >= 2009 & Weekly$Year <= 2010
X_train <- X[train_indices, ]
Y_train <- Y[train_indices]
X_test <- X[test_indices, ]
Y_test <- Y[test_indices]

evaluate_accuracy <- function(X_train, Y_train, X_test, Y_test, k) {
  knn_model <- knn(X_train, X_test, Y_train, k = k)
  accuracy <- sum(knn_model == Y_test) / length(Y_test)
  return(accuracy)
}

k_values <- seq(1, 20, by = 2)

max_accuracy <- 0
best_k <- NULL

for (k in k_values) {
  accuracy <- evaluate_accuracy(X_train, Y_train, X_test, Y_test, k)
  if (accuracy > max_accuracy) {
    max_accuracy <- accuracy
    best_k <- k
  }
}

knn_model <- knn(X_train, X_test, Y_train, k = best_k)

cat("Mejor valor de k:", best_k, "\n")
```

```
## Mejor valor de k: 15
```

```
cat("Precisión máxima en los datos de prueba:", max_accuracy, "\n")
```

```
## Precisión máxima en los datos de prueba: 0.5769231
```

En el código anterior (Cód.15) se carga la librería `class` para utilizar la función KNN. Luego, se define una función llamada `evaluate_accuracy` que evalúa la precisión del modelo KNN para un valor de k dado. Esta función toma los conjuntos de muestra y prueba, ajusta el modelo KNN con el valor de k proporcionado y calcula la precisión comparando las predicciones del modelo con las etiquetas reales en los datos de prueba. Se define un rango de valores de k impares a probar, en este caso, del 1 al 20. Se realiza una búsqueda exhaustiva sobre los valores de k. Para cada valor de k, se utiliza la función `evaluate_accuracy` para calcular la precisión

del modelo KNN. Se actualiza la variable `max_accuracy` con el valor de precisión más alto encontrado hasta el momento y se guarda el valor correspondiente de k en la variable `best_k`. Después de completar la búsqueda, se entrena un nuevo modelo KNN utilizando el mejor valor de k encontrado. Se evalúa la precisión del modelo en los datos de prueba y se imprime la precisión máxima obtenida y el valor de k correspondiente.

Con los resultados obtenidos (Cód.15) vemos que con un valor de $k = 15$ obtendremos una tasa de acierto del 57.69%. La misma que obtuvimos con el clasificador LDA. (Cód.10)

10. Support Vector Machines (Máquinas de Vector Soporte)

10.1 Introducción

Las SVM (Support Vector Machines) son un método de aprendizaje supervisado utilizado en problemas de clasificación y regresión. Desde su desarrollo, se han convertido en una herramienta fundamental en el campo del aprendizaje automático debido a su capacidad para abordar problemas lineales y no lineales de manera efectiva.

La idea central de las SVM es encontrar un hiperplano óptimo en un espacio dimensional superior que pueda separar dos clases de datos de manera óptima. Estos hiperplanos se eligen de tal manera que maximicen la distancia entre los puntos de datos más cercanos de cada clase, llamados vectores de soporte. Por lo tanto, las SVM buscan encontrar la mejor separación posible entre las clases, maximizando el margen entre ellas.

10.2 Hiperplano

En un espacio p-dimensional, un hiperplano es un subespacio plano y afín, no tiene por qué pasar por el origen, de dimensión p - 1. En dos dimensiones, un hiperplano es un subespacio plano unidimensional, lo cual se puede visualizar como una línea recta. En tres dimensiones, un hiperplano es un subespacio plano bidimensional, es decir, un plano. En dimensiones p > 3, puede resultar difícil visualizar un hiperplano, pero la idea de un subespacio plano de dimensión (p - 1) sigue siendo válida.

La definición matemática de un hiperplano es bastante simple. En el caso de dos dimensiones, un hiperplano se define conforme la ecuación de una recta

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

con parámetros β_0, β_1 y β_2. Para cualquier valor de $X = (X_1, X_2)^T$, los cuales cumplan la ecuación, serán considerados puntos en el hiperplano. La ecuación del hiperplano puede extenderse al contexto de p-dimensiones:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$

Si el punto $X_i = (x_1, x_2, \ldots, x_p)^T$ en el espacio de dimensión p es igual a 0 , tenemos que el punto X está en el hiperplano, pero si el punto X_i es mayor a 0,

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k > 0$$

X se sitúa a un lado del hiperplano y si es menor

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k < 0$$

estará al otro lado del hiperplano.

Podemos entender el hiperplano como una división del espacio de p-dimensional en dos partes. Para determinar en qué lado del hiperplano se encuentra un punto solo hay que calcular el signo del lado izquierdo de la ecuación.

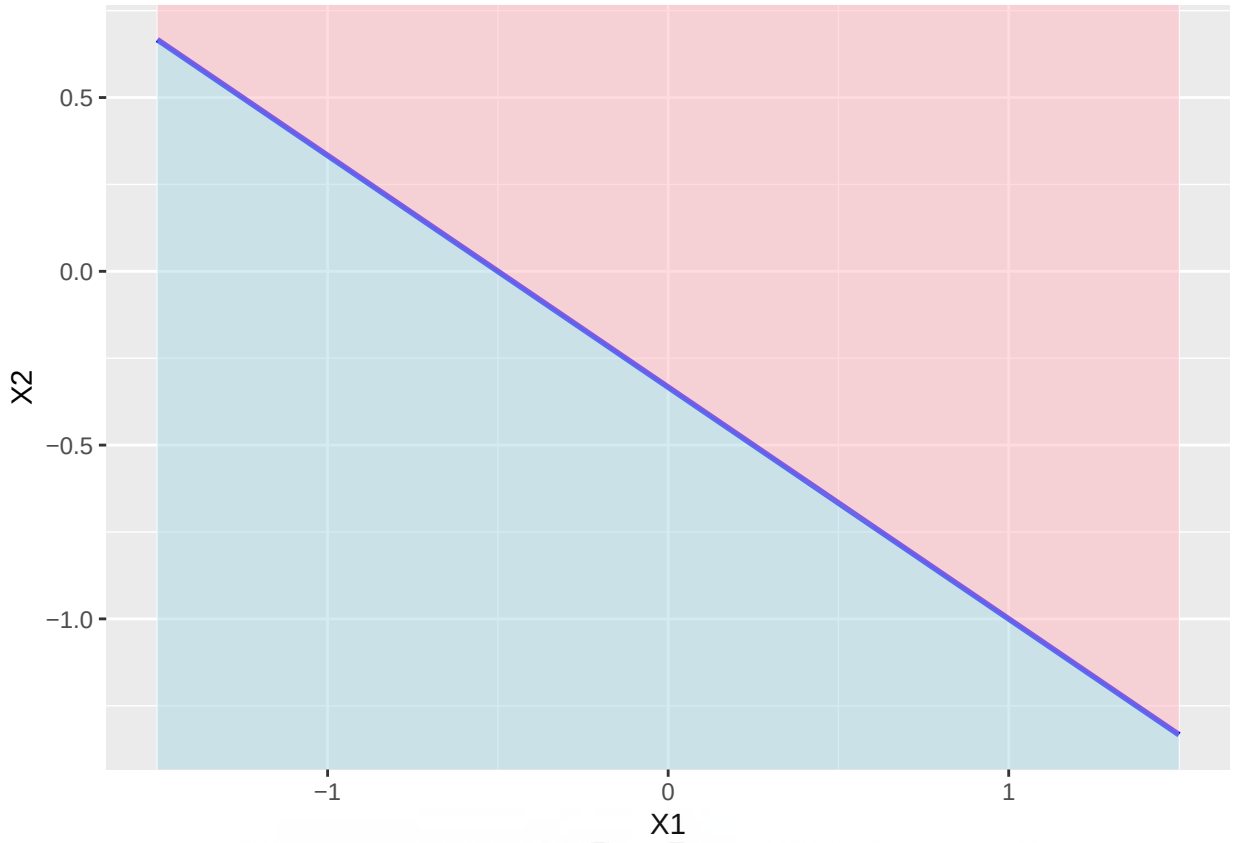


Figura 3. Separación del hiperplano $2X_1 + 3X_2 + 1 = 0$. La región rosada pertenece a aquellos puntos para los cuales el hiperplano $2X_1 + 3X_2 + 1 > 0$ y la región azul para el hiperplano $2X_1 + 3X_2 + 1 < 0$

10.3 Hiperplano de separación como clasificador

Supongamos que tenemos una matriz de datos X de tamaño $n \times p$ que consta de n observaciones de muestra en un espacio de p dimensiones y cuya variable respuesta se divide en dos clases, -1 y +1. También tenemos una observación de prueba, un vector de p características observadas $x^* = (x_1^* \dots x_p^*)^T$. El objetivo es desarrollar un clasificador basado en los datos de muestra que clasifique correctamente la observación de prueba utilizando sus características.

Si es posible construir un hiperplano que separe perfectamente las observaciones de muestra según sus etiquetas de clase, entonces tiene la propiedad de que

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} > 0 \quad si \quad y_i = 1$$

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} < 0 \quad si \quad y_i = -1$$

Como multiplicar dos números negativos es positivo, podemos definir un hiperplano de separación también como:

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) > 0 \quad para \quad i = 1, \dots, n$$

Si se cumple el método, podemos utilizarlo para construir un clasificador fácil de usar, asignando una observación de prueba a una clase según el lado del hiperplano en el que se encuentre. Es

decir, clasificamos la observación de prueba x^* en función del signo de $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \cdots + \beta_p x_p^*$. Si $f(x^*)$ es positiva, asignamos la observación de prueba a la clase 1, y si $f(x^*)$ es negativa, la asignamos a la clase -1. Además, si la magnitud de $f(x^*)$ está lejos del 0, entonces x^* se encuentra cerca del hiperplano, si por el contrario se encuentra cerca del 0, entonces esteremos menos seguros de la asignación de clase para x^*.

10.4 Clasificador de máximo margen

Si los datos pueden ser perfectamente separados utilizando un hiperplano, existirá un número infinito de hiperplanos que cumplen esta condición, lo que hace necesario un método que permita seleccionar uno de ellos como clasificador óptimo.

Construimos el hiperplano de margen máximo basado en un conjunto de n observaciones de muestra $x_1, \ldots, x_n \in \mathbb{R}^p$ y las etiquetas de clase asociadas $y_1, \ldots, y_n \in \{-1, 1\}$. El hiperplano de margen máximo es la solución al problema de optimización:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,M}{Maximizar} M$$

$$sujeto\ a \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} \geq M \quad \forall i = 1, \ldots, n$$

Las restricciones aseguran que cada observación esté en el lado correcto del hiperplano y al menos a una distancia M del hiperplano. Por lo tanto, M representa el margen de nuestro hiperplano, y el problema de optimización elige $\beta_0, \beta_1, \ldots, \beta_p$ para maximizar M.

10.5 Clasificador de vectores de soporte

El clasificador de vectores de soporte o clasificador de margen blando, a diferencia del clasificador de máximo margen, no busca el margen más grande posible para que cada observación esté en el lado correcto del hiperplano y en el lado correcto del margen, sino que permite que algunas observaciones estén en el lado incorrecto del margen, o incluso en el lado incorrecto del hiperplano.

El clasificador de vectores de soporte, o clasificador de margen blando, clasifica una observación de prueba según el lado del hiperplano en el que se encuentra. El hiperplano se elige para separar correctamente la mayoría de las observaciones de muestra en las dos clases, pero permitiendo clasificar erróneamente algunas observaciones. Definimos el clasificador de vectores de soporte como la solución al problema de optimización:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,M}{Maximizar} M$$

$$sujeto\ a \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C,$$

donde:

- M es la anchura del margen.
- C es un parámetro de ajuste no negativo, cuando C crece nos volvemos menos tolerantes a las violaciones del margen y por lo tanto el margen se estrecha, y si C decrece, el margen se ensanchará y por lo tanto, tendremos muchos vectores de soporte.
- ϵ_i nos indica donde se encuentra la i-ésima observación en relación al hiperplano y al margen. Si $\epsilon_i = 0$, entonces la i-ésima observación está en el lado correcto del margen, si $\epsilon_i > 0$ estará en el lado incorrecto del margen y si $\epsilon_i > 1$, entonces está en el lado incorrecto del hiperplano.

Una vez resuelto el problema clasificamos las observaciones de prueba determinando en qué lado del hiperplano se encuentra. Es decir, clasificamos la observación de prueba en función del signo de $f(x^*)$.

10.6 Clasificación con fronteras de decisión no lineales

A partir del clasificador de vectores de soporte podremos abordar el problema de las fronteras no lineales entre las clases, ampliando el espacio de características mediante el uso de funciones polinómicas cuadráticas, cúbicas e incluso de orden superior de los predictores. Por ejemplo, en lugar de ajustar un clasificador de vectores de soporte utilizando p características $X_1, X_2, \ldots, X_p$, lo ajustaremos utilizando 2p características $X_1, X_1^2, X_2, X_2^2, \ldots, X_p, X_p^2$.

El problema de optimización planteado para el clasificador de vectores de soporte ajustado al problema de fronteras de decisión no lineales quedaría así:

$$\underset{\beta_0,\beta_{1,1},\beta_{1,2},\ldots,\beta_{p,1},\beta_{p,2},\epsilon_1,\ldots,\epsilon_n,M}{Maximizar} M$$

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

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

Aunque la frontera de decisión que resulta del problema es lineal en el espacio de características original, la frontera de decisión tiene la forma $q(x) = 0$, donde q es un polinomio cuadrático, y sus soluciones normalmente son no lineales.

10.6 Kernels

A partir del clasificador de vectores de soporte abordamos el problema de fronteras no lineales con el uso de kernels, ampliando nuestro espacio de características para acomodar un límite no lineal entre las clases.

La idea se basa en el hecho de que la solución del problema del vector de soporte contiene únicamente el producto interior de las observaciones:

$$\langle x_i, x_i^{'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i^{'}j}$$

Se puede demostrar que el SVM puede ser representado como

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x_i, x_i^{'} \rangle,$$

siendo α_i parámetros que son distintos de cero solo para los vectores de soporte en la solución.

Al representar el clasificador lineal $f(x)$ y calcular sus coeficientes, todo lo que necesitamos son productos interiores. En SVM podemos reemplazar los productos interiores por una generalización de dicho producto (kernel), es decir, por $K(x_i, x_{i^{'}})$. Algunos ejemplos de kernel son:

- Kernel lineal: Cuantifica la similitud de un par de observaciones usando la correlación de Pearson (estándar). Nos devolvería el clasificador de vectores de soporte.

$$K(x_i, x_i^{'}) = \sum_{j=1}^{p} x_{ij} x_{i^{'}j}$$

- Kernel polinomial de grado d: Usar un kernel con d > 1, equivale a ajustar un vector de soporte clasificador en un espacio de mayor dimensión que involucra polinomios de grado d, en lugar del espacio de características original.

$$K(x_i, x_i^{'}) = \Big(1 + \sum_{j=1}^{p} x_{ij} x_{i^{'}j}\Big)^d$$

- Kernel radial: Si una observación de prueba dada x^* está lejos de una observación de entrenamiento x_i en términos de distancia euclídea, entonces $\sum_{j=1}^{k}(x_j^*)$ será grande, y por lo tanto $K(x_i, x_i^{'}) = exp\Big(-\gamma \sum_{j=1}^{p} \Big(x_{ij} - x_{i^{'}j}\Big)^2\Big)$ será muy pequeño, lo que significa que x_i no tomará ningún rol en $f(x^*)$.

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

10.7 SVM con más de dos clases

Aunque el concepto de hiperplanos separadores en el que se basan las SVM no se presta naturalmente a más de dos clases, se han propuesto varias alternativas para extender las SVM al caso de K clases, las dos más populares son la clasificación 1 contra 1 y la clasificación 1 contra todos.

10.7.1 Clasificación 1 contra 1

Supongamos que deseamos realizar la clasificación utilizando SVM y hay K $>$ 2 clases. El enfoque de uno contra uno o de todos los pares construye $\binom{k}{2}$ SVMs, cada una de las cuales compara un par de clases. Por ejemplo, una de esas SVMs podría comparar la k-ésima clase, codificada como +1, con la clase, con la k´-ésima clase, codificada como -1. Clasificamos una observación de prueba utilizando cada uno de los $\binom{k}{2}$ clasificadores, y contamos el número de veces que la observación de prueba se asigna a cada una de las K clases. La clasificación final se realiza asignando la observación de prueba a la clase a la que se le haya asignado con mayor frecuencia.

10.7.2 Clasificación 1 contra todos

Ajustamos K SVMs, comparando cada vez una de las K clases con las K-1 clases restantes. Sea $\beta_{0k}, \beta_{1k}, ..., \beta_{pk}$ los parámetros que resultan de ajustar una SVM comparando la k-ésima clase (codificada como +1) con las demás (codificadas como -1). Sea x^* una observación de prueba. Asignamos la observación a la clase para la cual $\beta_{0k} + \beta_{1k}x_1^* + \beta_{2k}x_2^* + \cdots + \beta_{1p}x_p^*$ es el valor más grande, ya que esto representa un alto nivel de confianza de que la observación de prueba pertenece a la k-ésima clase en lugar de a cualquiera de las otras clases.

10.8 Ventajas

1. Eficacia en espacios de alta dimensión: Los SVM funcionan bien incluso en espacios de características de alta dimensión, como aquellos en los que el número de características es mayor que el número de muestras. Esto se debe a que los SVM se basan en la idea de encontrar un hiperplano óptimo que separe las clases.
2. Buen rendimiento en conjuntos de datos pequeños: Los SVM son eficaces cuando se tiene un número limitado de muestras. Esto se debe a que los SVM intentan maximizar el margen entre las clases, lo que puede resultar en una mejor generalización en conjuntos de datos pequeños.
3. Flexibilidad en la elección de la función del kernel: Los SVM utilizan una función del kernel para mapear los datos a un espacio de características de mayor dimensión, lo que les permite manejar datos no linealmente separables. Existen diferentes funciones de kernel (lineal, polinómico, radial, entre otras), lo que brinda flexibilidad para adaptarse a diferentes tipos de datos.
4. Control de sobreajuste: Los SVM tienen parámetros que permiten controlar la complejidad del modelo y evitar el sobreajuste. Esto se logra a través del parámetro de regularización y la selección adecuada del kernel.

10.9 Desventajas

1. Sensibilidad a la escala de las características: Los SVM pueden verse afectados por la escala de las características. Si las características tienen diferentes escalas, es necesario realizar una normalización para que todas las características contribuyan de manera equitativa al modelo.
2. Dificultad en la elección del parámetro de regularización y del kernel: La elección adecuada del parámetro de regularización (C) y del kernel puede ser un desafío. Un valor

inapropiado de C puede conducir a un sobreajuste o subajuste del modelo. Además, la elección del kernel depende de la naturaleza de los datos y puede requerir pruebas y ajustes.

3. Ineficiencia computacional en conjuntos de datos grandes: El tiempo de estimación de parámetros de un SVM puede aumentar significativamente con conjuntos de datos grandes. Esto se debe a que el método debe considerar todos los vectores de soporte, lo que puede resultar en un alto costo computacional.
4. Dificultad para manejar conjuntos de datos desbalanceados: Los SVM pueden tener dificultades para lidiar con conjuntos de datos en los que una clase está dominada por otra. Esto se debe a que los SVM buscan maximizar el margen entre las clases y pueden verse sesgados hacia la clase dominante.

10.10 Programación en R

Ahora, realizaremos un SVM a los datos de `Weekly`. Para ello, utilizaremos la librería `e1071` la misma que utilizamos para resolver Naive Bayes.

Comenzaremos realizando un SVM con un kernel lineal para comprobar si los datos se distribuyen de manera lineal. Para buscar directamente el costo que mejor porcentaje de acierto nos proporcione, haremos similar que en KNN para encontrar el mejor k (Cód.15), realizando un algoritmo iterativo que pruebe distintos valores de costo.

```
#16
library(e1071)
costos <- c(0.001 ,0.01 ,0.1, 1, 10)

porcentaje_acierto <- numeric(length(costos))

for (i in 1:length(costos)) {

  svmModel <- svm(Direction ~ Lag1 + Lag2, data = train_Weekly, kernel = "linear", cost
  predictions <- predict(svmModel, test_Weekly)

  aciertos <- sum(predictions == test_Weekly$Direction)
  porcentaje_acierto[i] <- aciertos / length(predictions) * 100
}

plot(costos, porcentaje_acierto, type = "l", xlab = "Costo", ylab = "Porcentaje de Acier
```

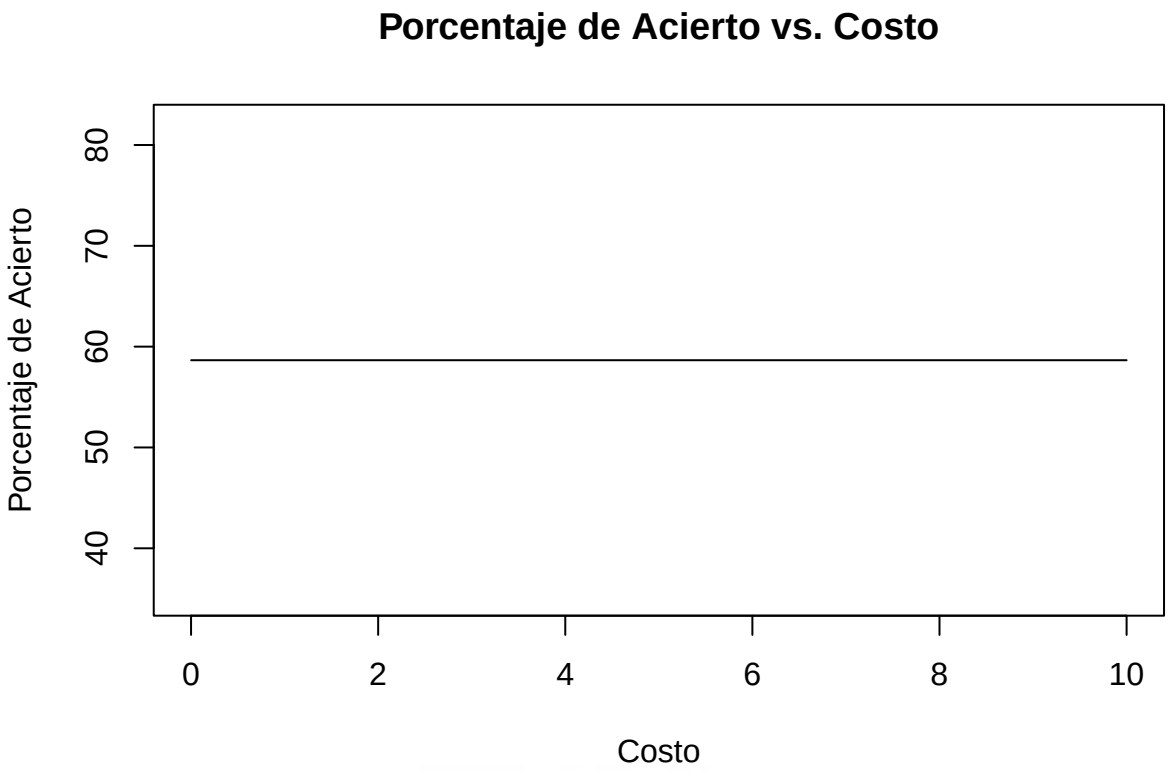


Figura 4. Gráfico que muestra el porcentaje de acierto en función del costo.

Como podemos ver (Fig.4) el porcentaje de acierto se mantendrá constante independientemente del costo seleccionado. Así que no hará falta especificar el costo en la función `svm` y por defecto será 1.

```
#17
svmModel <- svm(Direction ~ Lag1 + Lag2, data = train_Weekly, kernel = "linear")
predictions <- predict(svmModel, test_Weekly)
confusionMatrix(predictions, test_Weekly$Direction)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Down Up
##       Down    0  0
##       Up     43 61
##
##                Accuracy : 0.5865
##                  95% CI : (0.4858, 0.6823)
##     No Information Rate : 0.5865
##     P-Value [Acc > NIR] : 0.5419
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 1.504e-10
##
##             Sensitivity : 0.0000
```

```
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.5865
##              Prevalence : 0.4135
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : Down
##
```

Si nos fijamos (Cód.17) la estructura es parecida a los métodos anteriores: estimamos los parámetros, predecimos sobre los datos de prueba y validamos. En este caso tendríamos un 58.65% de acierto. Pero cuidado, si nos fijamos, SVM ha predicho para todas las observaciones de prueba que los rendimientos van a subir, aunque el porcentaje de acierto es mayor a los otros métodos, tendremos que fijarnos en algo más. El índice de Kappa es igual a 0, esto que quiere decir, que quizás tengamos alguno de estos problemas: un problema de sobreajuste, las características no son adecuadas para el modelo o el tamaño o la calidad del conjunto de muestra no son adecuados.

En resumen, si el índice kappa es 0, es una señal de que el modelo no está funcionando adecuadamente y se deben explorar diferentes enfoques para mejorar su rendimiento.

Ahora, resolveremos SVM con un kernel radial. Lo primero será graficar para visualizar cuales serán los mejores valores para costo y gamma mediante validación cruzada.

```r
#18
costos <- seq(0.0001, 3, by = 0.1)
gammas <- c(0.1, 1, 5, 10)
porcentaje_acierto <- matrix(0, nrow = length(costos), ncol = length(gammas))

for (i in 1:length(costos)) {
  for (j in 1:length(gammas)) {
    svmModel <- svm(Direction ~ Lag1 + Lag2, data = train_Weekly, kernel = "radial", cos
    predictions <- predict(svmModel, test_Weekly)

    aciertos <- sum(predictions == test_Weekly$Direction)
    porcentaje_acierto[i, j] <- aciertos / length(predictions) * 100
  }
}

plot(costos, porcentaje_acierto[,1], type = "l", xlab = "Costo", ylab = "Porcentaje de A
lines(costos, porcentaje_acierto[,2], type = "l", col = "red")
lines(costos, porcentaje_acierto[,3], type = "l", col = "green")
lines(costos, porcentaje_acierto[,4], type = "l", col = "purple")
legend("topright", legend = gammas, col = c("blue", "red", "green", "purple"), lty = 1,
```

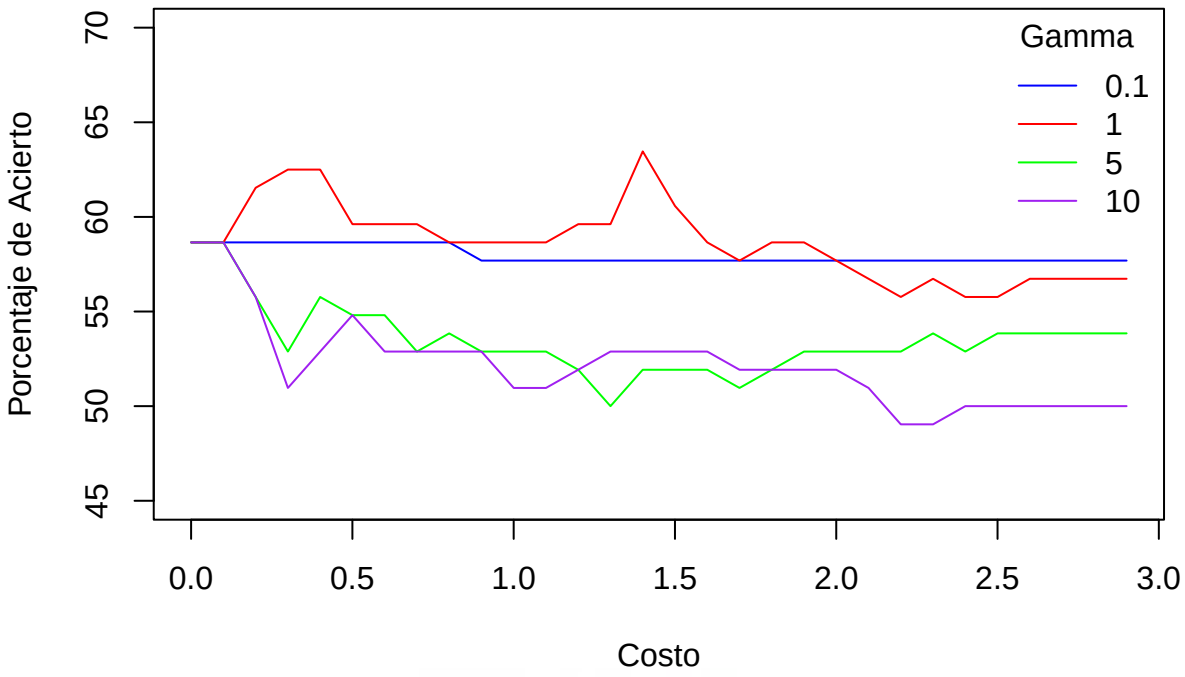


Figura 5. Gráfico que muestra el porcentaje de acierto en función del costo y el gamma.

Como podemos ver, en la Figura 5 la mejor precisión será con un gamma de 1 y un costo aproximado a 1.5. Si visualmente no está claro, también podemos calcular el punto óptimo:

```
#19
max_porc_acierto <- max(porcentaje_acierto)
optimo <- which(porcentaje_acierto == max_porc_acierto, arr.ind = TRUE)
optimo_costo <- costos[optimo[1,1]]
optimo_gamma <- gammas[optimo[1,2]]

cat("Punto óptimo:\n")
```

```
## Punto óptimo:
```

```
cat("Costo:", optimo_costo, "\n")
```

```
## Costo: 1.4001
```

```
cat("Gamma:", optimo_gamma, "\n")
```

```
## Gamma: 1
```

```
cat("Porcentaje de acierto:", max_porc_acierto, "\n")
```

```
## Porcentaje de acierto: 63.46154
```

Pues con un acierto del 63.46%, SVM es el clasificador que mejor porcentaje de acierto tiene para la clasificación de los rendimientos porcentuales semanales propuesto en la base de datos. Pero antes de darlo por ganador comprobaremos su índice Kappa.

```r
#20
svmModel <- svm(Direction ~ Lag1 + Lag2, data = train_Weekly, kernel = "radial", cost=1.
predictions <- predict(svmModel, test_Weekly)
confusionMatrix(predictions, test_Weekly$Direction)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Down Up
##       Down   18 13
##       Up     25 48
##
##                Accuracy : 0.6346
##                  95% CI : (0.5345, 0.7269)
##     No Information Rate : 0.5865
##     P-Value [Acc > NIR] : 0.18545
##
##                   Kappa : 0.2143
##
##  Mcnemar's Test P-Value : 0.07435
##
##             Sensitivity : 0.4186
##             Specificity : 0.7869
##          Pos Pred Value : 0.5806
##          Neg Pred Value : 0.6575
##              Prevalence : 0.4135
##          Detection Rate : 0.1731
##    Detection Prevalence : 0.2981
##       Balanced Accuracy : 0.6027
##
##        'Positive' Class : Down
##
```

Analizando el índice kappa, que es de 0.2143, se puede interpretar una concordancia baja o pobre. Esto significa que hay una discrepancia significativa entre las clasificaciones observadas y las esperadas por azar, lo que indica una capacidad limitada del SVM utilizado para clasificar los datos correctamente.

Es importante tener en cuenta que el valor del índice kappa depende del problema y los datos específicos. Un valor de 0.2143 puede ser considerado aceptable en ciertos contextos, mientras que en otros puede indicar la necesidad de mejorar o ajustar el modelo de clasificación utilizado.

11. Conclusiones

En conclusión, cada método de clasificación se basa en diferentes conceptos teóricos al aplicarse a la predicción de variables cualitativas.

Regresión logística se basa en la regresión y utiliza una función logística para predecir las probabilidades de las clases, mientras que LDA y QDA se enfocan en la reducción de dimensionalidad y el modelado de covarianzas para asignar clases.

LDA busca encontrar una combinación lineal de variables predictoras que maximice la separación entre las clases, mientras que QDA permite diferentes covarianzas entre las clases.

Naive Bayes se basa en el teorema de Bayes y asume independencia condicional entre las variables predictoras, calculando las probabilidades condicionales para asignar clases.

KNN clasifica las observaciones en función de la clase mayoritaria de sus vecinos más cercanos. No hace suposiciones específicas sobre la distribución de los datos y asigna una clase en función de la mayoría de los vecinos cercanos.

SVM busca encontrar un hiperplano de separación óptimo, maximizando el margen entre las observaciones de muestra más cercanas y manejando relaciones no lineales mediante el uso de funciones de kernel.

Basándonos en los resultados del estudio, podemos obtener las siguientes conclusiones:

SVM muestra el mejor rendimiento con una precisión del 63.46%. Esto indica que SVM fue capaz de capturar patrones y tendencias en los datos del índice bursátil y predecir su rendimiento semanal con mayor precisión en comparación con los otros métodos.

Tanto LDA como KNN también obtuvieron buenos resultados con una precisión del 57.69%. Estos métodos mostraron ser eficaces para predecir el rendimiento semanal del índice bursátil, aunque ligeramente menos precisos que SVM.

Regresión logística, Análisis Discriminante Cuadrático y Naive Bayes obtuvieron resultados similares, con una precisión alrededor del 54-55%. Aunque estos métodos no lograron el rendimiento más alto, aún pueden proporcionar predicciones aceptables para el rendimiento semanal del índice bursátil.

Es importante tener en cuenta que la predicción del rendimiento del mercado financiero es un desafío debido a la naturaleza volátil y no lineal de los datos. Estos resultados indican que los métodos de clasificación utilizados pueden capturar ciertos patrones y tendencias en el comportamiento del índice bursátil, pero no garantizan una predicción precisa en todos los casos. Además, es fundamental tener en cuenta que los resultados obtenidos se basan en un conjunto de datos específico y pueden no ser generalizables a otros índices bursátiles o periodos de tiempo. Por lo tanto, se recomienda realizar más investigaciones y análisis antes de tomar decisiones financieras basadas únicamente en estos resultados.

Método de clasificación	Porcentaje de éxito
Regresión Logística	55.77%
Análisis Discriminante Lineal	57.69%
Análisis Discriminante Cuadrático	55.77%
Naive Bayes	53.85%
K-Nearest Neighbors	57.69%
Support Vector Machines	63.46%

12. Bibliografía

1. Gareth James, Daniela Witten, Trevor Hastie & Robert Tibshirani (2021). *An Introduction to Statistical Learning with Applications in R*, 2nd Edition. Springer.
2. Trevor Hastie, Robert Tibshirani, Jerome Friedman (2017). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd Edition. Springer.
3. Domingo Morales Gonzalez (2022). *Estadística Avanzada*. Universidad Miguel Hernández de Elche.
4. Marina Leal (2022). *Técnicas Estadísticas en Análisis de Mercados*. Universidad Miguel Hernández de Elche.
5. R Core Team (2023) *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna. https://www.R-project.org.
6. RStudio Team (2021). *RStudio: Integrated Development Environment for R*. RStudio, PBC, Boston, MA. http://www.rstudio.com/.
7. MiKTeX. https://miktex.org/.
8. Allaire J, Xie Y, Dervieux C, McPherson J, Luraschi J, Ushey K, Atkins A, Wickham H, Cheng J, Chang W, Iannone R (2023). *rmarkdown: Dynamic Documents for R*. https://github.com/rstudio/rmarkdown.
9. Gareth James, Daniela Witten, Trevor Hastie, Rob Tibshirani (2022). *ISLR2: Introduction to Statistical Learning, Second Edition*. https://CRAN.R-project.org/package=ISLR2.
10. Brian Ripley, Bill Venables, Douglas M. Bates, Kurt Hornik, Albrecht Gebhardt, David Firth (2023). *MASS: Support Functions and Datasets for Venables and Ripley's MASS*. https://CRAN.R-project.org/package=MASS.
11. David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, Friedrich Leisch, Chih-Chung Chang, Chih-Chen Lin. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group*. https://CRAN.R-project.org/package=e1071.
12. Brian Ripley, William Venables. *class: Functions for Classification*. https://CRAN.R-project.org/package=class.
13. Hadley Wickham, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, Dewey Dunnington (2023). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. https://CRAN.R-project.org/package=ggplot2.
14. Barret Schloerke, Di Cook, Joseph Larmarange, Francois Briatte, Moritz Marbach, Edwin Thoen, Amos Elberg, Ott Toomet, Jason Crowley, Heike Hofmann, Hadley Wickham. *GGally: Extension to 'ggplot2'*. https://CRAN.R-project.org/package=GGally.
15. Max Kuhn ORCID, Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer, Brenton Kenkel, R Core Team, Michael Benesty, Reynald Lescarbeau, Andrew Ziem, Luca Scrucca, Yuan Tang, Can Candan, Tyler Hunt. *caret: Classification and Regression Training*. https://CRAN.R-project.org/package=caret.