Predicción de resultados de partidos de fútbol

Sergio Sabroso Lasa

Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Director del trabajo: Javier López Lorente
28 de junio de 2018

Abstract

This project consists of three chapters and we are going to summarize here the most important ideas.

There are many models for predicting the outcome of fooball matches, but we are going to build two logistic models, one of them to predict the probability of the event "local team wins" and another to predict the probability of the event "away team wins". For that, we use simple predictor variables such as the point difference between local team and away team or how the teams performed in the last matches. In our first model, the probability α of the event "local team wins" in a football match follows the equation:

$$\alpha = \frac{e^{-0,3+0,173x_2+0,083x_5}}{1+e^{-0,3+0,173x_2+0,083x_5}}$$

where X_2 is the predictor variable which represents the point difference divided by the logarithm of the match day and X_5 is the predictor variable "local run" which summarizes the last results of the local team. On the other hand, the probability γ of the event "away team wins" follows the equation:

$$\gamma = \frac{e^{-1,04-0,187x_2}}{1+e^{-1,04-0,187x_2}}$$

where X_2 is the same as before. Then we study the results using ROC curves and we confirm that these methods predicts well the result of a football match.

Once we have two models to predict football results we want to use them for betting. In that aim, we introduce a new betting strategy. First we review the classic Kelly´s criterion and we see that it´s inappropiate because we lose all our money in a few bets. Then we design a dynamic programming problem which decides, for each match day, how much money to bet in each match in order to maximize the probability of reaching a money target. We program this problem in *Java* and apply it to a real life situation, namely, the Santander Liga $2017-2018$ where we obtain a benefit of $600\,\%$.

Índice general

Capítulo 1

Introducción

No cabe ninguna duda de que el fútbol es el deporte más popular y seguido del mundo, basta comprobar cómo la gente está pegada al televisor en sus casas, bares y restaurantes cuando se disputa un partido importante. Para hacernos una idea de la popularidad de este deporte es suficiente ver, como se explica en [12], que la final del mundial de 2014 tuvo alrededor de 1000 millones de espectadores, comparado con la inauguración de los Juegos Olímpicos en Londres 2012 con 900 millones o la final de los playoffs de la NBA del 2017 con unos 20 millones. La idea de este deporte es muy sencilla, y hoy en día es casi imposible encontrar a alguien que no tenga una noción básica de cómo se practica. El ganador de un partido es el equipo que más goles marca en un periodo de 90 minutos dividido en dos partes.

El fútbol comenzó en Inglaterra con partidos casi sin normas que se practicaban en cualquier sitio, como plazas, calles... Pero ya en aquel entonces los ingleses mezclaron el deporte con la economía, en el sentido de que la gente pagaba una cantidad de dinero por ver un partido de los equipos más populares. Este deporte fue evolucionando hasta que entre 1850 y 1890 se introdujeron las reglas definitivas que hoy en día conocemos. Actualmente, el fútbol mueve tanta cantidad de dinero que hay mucha gente que lo considera un negocio, y otra que realmente hace un negocio de él mediante apuestas deportivas.

Las apuestas deportivas no son tan recientes como se cree, sino que surgieron hace más de 2000 años en los Juegos Olímpicos que se disputaban en Grecia. Se hicieron mucho más populares con las carreras de caballos durante el siglo XIX, pero su gran expansión no fue hasta 1940 cuando aparecieron las primeras casas de apuestas en Nevada (EEUU), permitiendo apostar a diferentes modalidades deportivas. El auge de las apuestas deportivas tuvo lugar en 1996 con Internet, ya que se lanzó *Intertops*, la primera página que permitía apostar online. Gracias a este sistema, los usuarios pueden apostar desde casa mientras están viendo los eventos, tomando decisiones al momento y apostando de forma directa y rápida en segundos.

Desde su comienzo, la popularidad de las apuestas ha ido creciendo hasta tal punto que en el año 2017, de acuerdo con [13], se movieron más de 5000 millones de euros sólo en España. Por esta razón, poder predecir el resultado de un partido de fútbol mediante variables intuitivas y sencillas, es una idea ambiciosa cuya finalidad busca obtener beneficio económico mediante su aplicación a apuestas deportivas.

Existen gran cantidad de portales web en los que los resultados de temporadas anteriores están accesibles, incluso se puede comprobar casi cualquier dato sobre los partidos ya disputados; desde el resultado final, hasta la cantidad de jugadores lesionados que tenía un equipo antes de comenzar el partido. Por tanto se puede aprovechar toda esta información almacenada para crear modelos que predigan el resultado de un cierto partido, aunque ya existen varios modelos capaces de dar unas probabilidades muy cercanas a la realidad si se dispone de los datos suficientes. Un modelo popular y que ha sido

revisado varias veces es el de Maher (1982), en el que se intenta predecir el número de goles que cada equipo anota mediante variables Poisson multivariantes.

Nuestro trabajo consta de dos partes diferenciadas. En primer lugar estudiaremos partidos de fútbol e intentaremos predecir sus resultados. En estos pueden darse tres resultados distintos: *victoria local*, *victoria visitante* y *empate*, pero sólo modelaremos dos de los casos, el de victoria local y el de victoria visitante. El estudio lo haremos mediante regresión logística múltiple con variables predictoras sencillas. Además, queremos obtener modelos aplicables a cualquier competición y que sean capaces de predecir solamente por cómo están jugando en la competición actual, sin tener en cuenta el "nombre" de los equipos que disputan el partido.

Una vez que tengamos las probabilidades de nuestros modelos, la segunda parte consistirá en buscar una estrategia óptima para obtener el mayor beneficio posible en un caso real de apuestas. Para esta parte del trabajo es necesario repasar el concepto básico de *cuota*. La *cuota* es la cantidad ofrecida por una casa de apuestas que multiplicará nuestra inversión en caso de acertar la apuesta. Como es lógico, la cuota nunca es más pequeña que 1, y en el caso de que una casa de apuestas de una cuota muy alta para un suceso es debido a que considera que tiene una probabilidad muy baja de ocurrir.

El objetivo es obtener beneficio apostando jornada a jornada de la temporada, por tanto tenemos que diseñar una estrategia óptima que nos indique a qué apostar y qué cantidad. Para ello en primer lugar repasaremos el criterio clásico de Kelly y veremos si lo podemos aplicar a nuestro problema, y en segundo lugar diseñaremos un problema de programación dinámica que nos maximice la probabilidad de llegar a una cierta cantidad de dinero en un número determinado de jornadas. Dado que en nuestro problema de programación dinámica estocástica hacen falta una gran cantidad de variables y cálculos, diseñaremos en *Java* un programa que sea capaz de resolverlo computacionalmente.

La parte final del trabajo consistirá en aplicar todos los resultados obtenidos a la temporada 2017-2018 de la Liga Santander, comprobar si obtenemos beneficios y confirmar así que nuestro modelo de predicción y la estrategia de apuestas seguida, son buenos y aplicables a la vida real.

Capítulo 2

Modelo de regresión logística

2.1. Introducción

Estamos interesados en encontrar un procedimiento para estimar la probabilidad α de que un equipo (por ejemplo el local) obtenga la victoria. Desarrollaremos dos modelos, uno para obtener la probabilidad de victoria del equipo local y otro para la probabilidad de victoria del equipo visitante.

Vamos a considerar Y como la variable aleatoria *victoria equipo local*, que toma los valores:

- 1 si el equipo local gana.
- 0 en cualquier otro caso (empata o pierde).

En contraposición a los modelos de regresión vistos en la asignatura del grado, donde la variable respuesta Y ha sido considerada como una variable continua y cuantitativa, en nuestra situación la variable respuesta es cualitativa. Para resolver estas situaciones hay varios métodos distintos del de mínimos cuadrados. Primero repasemos el concepto de variable dicotómica.

Definición 1. Una variable aleatoria X se dice dicotómica si solo puede tomar dos valores, normalmente representados con 1 como éxito y 0 como fracaso.

Por tanto, es claro que la variable Y mencionada anteriormente es dicotómica.

Estas situaciones donde la variable respuesta es dicotómica son bastante comunes y ocurren con frecuencia en aplicaciones estadísticas. Por ejemplo, si una persona tiene o no una determinada enfermedad, o si una empresa después de una determinada inversión permanece solvente o entra en bancarrota, son casos en los que la variable respuesta puede ser codificada con los valores 0 y 1. Para predecir la variable respuesta en situaciones de este estilo, se intentan modelar las probabilidades que toma Y para uno de esos dos valores cuando conocemos las variables predcitoras. En nuestro caso nos centraremos en obtener $P(Y = 1)$.

El modelo de regresión lineal por mínimos cuadrados presenta varias inconsistencias cuando la variable respuesta es dicotómica. Vamos a ilustrar ésto considerando un problema de regresión lineal simple en el que sólo tenemos una variable predictora (los resultados obtenidos son válidos también para el caso de regresión múltiple).

Vamos a denotar α como la probabilidad de que $Y = 1$ cuando $X = x$. Si usamos el modelo estándar de regresión lineal para precedir α, obtenemos lo siguiente:

$$\alpha = P(Y = 1|X = x) = \beta_0 + \beta_1 x + \varepsilon \tag{2.1}$$

Como α es una probabilidad debe tomar valores entre 0 y 1, pero es claro que la función lineal dada en (2.1) puede tomar valores fuera del intervalo $[0, 1]$, y por lo tanto este modelo no puede ser utilizado para predecir probabilidades. Otra razón por la que el método ordinario de mínimos cuadrados es inapropiado es debido a que la variable respuesta Y es una variable aleatoria Bernoulli, y como consecuencia, su

varianza depende de α, por lo que no se cumple la hipótesis de varianza constante (homocedasticidad) que es necesaria para realizar un ajuste de regresión lineal por mínimos cuadrados.

Por esta razón, el análisis de modelos de regresión con variable respuesta dicotómica es muy distinto al de mínimos cuadrados ordinarios.

2.2. Modelo logit

En la mayoría de las situaciones donde la variable respuesta es dicotómica, la relación entre la probabilidad α y la variable predictora X parece una curva con forma de S, como se puede apreciar en la Figura 2.1, conocida como *S-Shaped curve*.

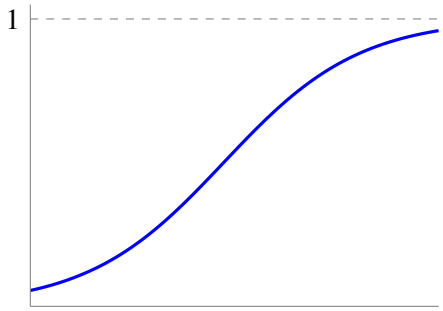


Figura 2.1: S-Shaped curve

Vemos que inicialmente la probabilidad α aumenta lentamente a medida que se incrementa X, posteriormente este incremento es mayor, y finalmente se estabiliza; pero no va más allá de 1, lo que intuitivamente tiene sentido ya que estamos hablando de probabilidades.

La forma de la *S-Curve* puede ser reproducida si modelizamos las probabilidades de la siguiente forma:

$$\alpha = P(Y = 1|X = x) = \frac{e^{\beta_0+\beta_1 x}}{1 + e^{\beta_0+\beta_1 x}} \tag{2.2}$$

Esto es lo que conocemos como modelo de regresión logística, aunque cabe destacar que hay otras formas de modelizar las probabilidades que también reproducen la *S-Curve*.

El modelo logistico puede ser genaralizado directamente a la situación en la que tenemos varias variables predictoras, en cuyo caso la probabilidad α viene dada por:

$$\alpha = P(Y = 1|X_1 = x_1, \ldots, X_p = x_p) = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_p x_p}}{1 + e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_p x_p}} \tag{2.3}$$

La ecuación (2.3) es llamada *función de regresión logística*. Como se puede apreciar, estamos ante una función no lineal de parámetros $\beta_i \; \forall i = 0, \ldots, p$, sin embargo, puede ser linealizada por medio de lo que se conoce como *transformación logistica*. En este caso, en lugar de trabajar directamente con α, se trabaja con un valor transformado de α.

En primer lugar, si α es la probabilidad de que un evento ocurra, la proporción $\frac{\alpha}{1-\alpha}$ se conoce como la proporción de probabilidades, *odds ratio* (cociente de probabilidades), para dicho evento. Como

$$1 - \alpha = P(Y = 0|X_1 = x_1, \ldots, X_p = x_p) = \frac{1}{1 + e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_p x_p}}$$

Se tiene que

$$\frac{\alpha}{1-\alpha} = e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_p x_p}$$

Tomando logaritmos en ambos lados, obtenemos

$$g(x_1, \ldots, x_p) = log\left(\frac{\alpha}{1-\alpha}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p \tag{2.4}$$

El logaritmo del *odds ratio* se conoce como *logit*. Por la ecuación (2.4) puede verse que la transformación logit produce una función lineal de parámetros $\beta_i \; \forall i = 0, \ldots, p$. Notar también que el rango de valores de α en (2.3) está entre 0 y 1, lo que implica que el rango de valores de $log\left(\frac{\alpha}{1-\alpha}\right)$ está entre $-\infty$ y ∞, por lo que es más apropiado para el ajuste de regresión lineal.

2.2.1. Obtención de los estimadores

En cuanto a la obtención de los estimadores el método usado es el de máxima verosimilitud, donde las estimaciones son calculadas numéricamente usando métodos iterativos. Por lo tanto, al contrario que en el método de mínimos cuadrados, no existe una expresión analítica sencilla para los estimadores de los parámetros así que no nos centraremos en los aspectos computacionales de como se obtienen.

2.3. Presentación de los modelos

Una vez explicado el modelo de regresión logística vamos a aplicarlo a nuestro problema, en el que queremos predecir la variable dicotómica *Y victoria equipo local* a partir de varias variables predictoras (regresoras). Cabe destacar que queremos diseñar un modelo que nos pueda predecir si gana un equipo o no sin importarnos la competición en la que juegue, ni el "nombre" de los equipos que disputen el partido, simplemente por cómo están jugando en el campeonato. Además, parece lógico que al tratarse de partidos de fútbol entre equipos de una misma clasificación, se tenga en cuenta variables como la posición en la tabla del equipo local, la posición en la tabla del equipo visitante, en que jornada del campeonato nos encontramos y alguna variable que nos informe de la forma física de cada equipo en el momento del partido. Por estos motivos, hemos decidido ensayar un modelo de regresión logística múltiple con las variables que se indican a continuación:

- X_1 representa la diferencia de puntos entre el equipo local y el visitante dividido por el número de la jornada en la que se disputa el partido (*DPJornada*).
- X_2 representa la diferencia de puntos entre el equipo local y el visitante dividido por el logaritmo del número de la jornada en la que se disputa el partido (*DPLogJornada*) (más adelante explicaremos el porqué de la elección de esta variable).
- X_3 representa lo que interpretaremos como racha del equipo que juega como local (*RL1*).
- X_4 representa la que interpretaremos como racha del equipo que juega como visitante (*RV1*).
- X_5 representa una definición alternativa de racha del equipo que juega como local (*RL2*).
- X_6 representa una definición alternativa de racha del equipo que juega como visitante (*RV2*).

Vamos a explicar brevemente lo que hemos entendido por racha. La racha de un equipo es el estado de forma en el que se encuentra en el momento de disputar un partido, coloquialmente hablando se dice lo "fuerte" que está un equipo, y como es lógico, un equipo está en mejor estado de forma (está más fuerte) conforme va ganando partidos consecutivos.

Por poner un ejemplo sencillo, imaginémonos que el *equipo A* lleva 3 partidos seguidos ganando y se encuentra décimo en la tabla y que por otro lado tenemos el *equipo B* que lleva 2 partidos seguidos perdiendo aunque se encuentra cuarto en la tabla. En este caso si sólo se tuviera en cuenta la posición de cada equipo en la tabla, parece lógico dar una probabilidad mayor de vencer al *equipo B*, pero la realidad no es así, ya que el *equipo A* está mucho más "fuerte" y puede llevarse perfectamente el partido.

Dicho esto, hemos creado la variable "racha" (variables *RL1* y *RV1*) para cada equipo de la siguiente forma:

- Es una variable cuyo rango está en $[1,5]$.

- Todos los equipos empiezan la temporada con una racha de 1.
- Si un equipo gana, entonces su racha aumentará en 1 en la jornada siguiente hasta alcanzar la racha máxima (5), la cual no puede aumentar más aunque el equipo gane.
- Si un equipo empata, entonces su racha de la jornada siguiente permanecerá igual.
- Si un equipo pierde, entonces se "rompe" su racha, y por lo tanto en la jornada siguiente tendrá valor de 1.

Alternativamente hemos definido las variables *RL2* y *RV2* de forma análoga con la única diferencia de que cuando un equipo pierde, su racha disminuye en 1 en lugar de volver a 1. Destacar que estas definiciones del concepto de racha son de nuestra invención y podría ser definido de muchas otras maneras diferentes.

A continuación veamos cómo hemos llegado a la elección de la variable *DPLogJornada*; en primer lugar, es obvio que para predecir Y va a influir tanto la diferencia de puntos entre ambos equipos, como la jornada en la que se dispute el partido ya que, dependiendo de la jornada en la que estemos, la diferencia de puntos puede ser más o menos significativa. Pongamos también un sencillo ejemplo; una diferencia de 8 puntos entre el equipo local y el visitante en la jornada 4 es mucho más significativa que una diferencia de 8 puntos en la jornada 34 (lo que querrá decir que son dos equipos muy cerca en la clasificación y que apenas hay diferencia entre ellos). Por lo tanto, queremos que en las primeras jornadas una diferencia de puntos pequeña sea bastante significativa, y conforme vaya aumentando el número de la jornada, una diferencia de puntos pequeña sea menos importante. Si hacemos solamente el cociente como en la variable *DPJornada* nos encontraremos con el problema de que la jornada tendrá mucho mas peso que la diferencia de puntos debido a que crece más rápido y eso tampoco es conveniente, pero si le introducimos el logaritmo (variable *DPLogJornada*), ya crece más despacio y se ajusta mejor a lo que queremos.

2.3.1. Creación y resultados del modelo *victoria local*

Para realizar el modelo, hemos creado e introducido en *R* una base de datos con todos los partidos de la primera división de Liga Española desde la temporada $2013-2014$ hasta la temporada $2016-2017$, lo que hacen un total de 1520 partidos. De estos partidos, hemos quitado aquellos en los que $Jornada < 5$, debido a que al principio todos los equipos empiezan con los mismos puntos y con la misma racha, y también aquellos en los que $Jornada > 35$, ya que al final de temporada muchos equipos bajan su nivel debido a que aunque ganen o pierdan tienen asegurada una cierta posición en la tabla. Así pues, parece aconsejable eliminarlos de la base de datos ya que en ambos casos son muy difíciles de predecir y obtendríamos datos anómalos. Para cada uno de los partidos restantes, hemos comprobado los puntos que llevaba cada equipo en la clasificación antes de comenzar éste, y hemos calculado la racha de cada uno en función del resultado de la jornada anterior. Posteriormente validaremos tanto este modelo como el de *victoria visitante* con esta base de datos y con los resultados de la temporada $2017-2018$.

Dado que tenemos un número elevado de variables predictoras y algunas muy parecidas, hemos realizado el modelo de regresión logística paso a paso con criterio *AIC*. El criterio *AIC* es una medida estadística que proporciona un modelo con las variables más significativas, en el que se pierda la menor cantidad de información posible (para ver las ordenes ejecutadas para llegar al modelo, consultar el script en el anexo). Los resultados obtenidos son los siguientes:

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.30090    0.13214  -2.277   0.0228 *
RL2           0.08345    0.04525   1.844   0.0651 .
DPLogJornada  0.17287    0.01689  10.233   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1772.3  on 1279  degrees of freedom
Residual deviance: 1562.3  on 1277  degrees of freedom
AIC: 1568.3

Number of Fisher Scoring iterations: 3
```

Por tanto el modelo de *victoria local* cuenta con variables predictoras *DPLogJornada* y *RL2*. Es bastante destacable el hecho de que ninguna de las variables que definen la racha visitante sean significativas a la hora de predecir si un equipo local gana o no, ya que el modelo sólo señala interesante el estado de racha del equipo local y la diferencia de puntos según en la jornada en la que se esté a la hora de predecir. La estimación de los parámetros es:

- $\hat{\beta}_0 = -0,301$
- $\hat{\beta}_2 = 0,173$
- $\hat{\beta}_5 = 0,083$

Estos coeficientes determinan que si aumentamos en una unidad la variable *DPLogJornada* o la variable *RL2*, el *odds ratio* se incrementa en $e^{0,173}$ ó $e^{0,083}$ respectivamente.

Así pues, la probabilidad α de que en un determinado partido, gane el equipo local, viene dada por:

$$\alpha = P(Y = 1|X_2 = x_2, X_5 = x_5) = \frac{e^{-0,301+0,173x_2+0,083x_5}}{1+e^{-0,301+0,173x_2+0,083x_5}}$$

Luego haciendo la *transformación logística* mencionada anteriormente tenemos que:

$$log\left(\frac{\alpha}{1-\alpha}\right) = -0,301+0,173x_2+0,083x_5$$

Viendo la salida por pantalla de *R*, se aprecia claramente que la variable más significativa y por tanto mas importante a la hora de predecir es *DPLogJornada* (es la que tiene un ρ-valor más pequeño). La relación entre esta variable y la probabilidad α viene dada por la Figura 2.2.

Como se aprecia, la Figura 2.2 también tiene una forma de *S-Curve*, luego parece haber sido apropiada la elección de un modelo de regresión logística para predecir la victoria del equipo local en un determinado partido de fútbol. Posteriormente validaremos más el modelo.

2.3.2. Creación y resultados del modelo *victoria visitante*

Para la creación de este modelo hemos seguido el mismo procedimiento que en el anterior, un modelo de regresión logística paso a paso con criterio *AIC* sobre la misma base de datos. Al realizar el modelo paso a paso, *R* nos intuía como mejor modelo uno con *DPLogJornada*, *DPJornada* y *RV1* como variables predictoras, pero no podemos aceptarlo como bueno dado que las variables *DPLogJornada* y *DPJornada* están muy correladas (coeficiente de correlación de Pearson de $0,95$). Esta situación llamada multicolinealidad aproximada, implica que las varianzas de los estimadores, y por lo tanto las predicciones, sean grandes e imprecisas. Para solventar este problema hemos planteado las siguientes alternativas:

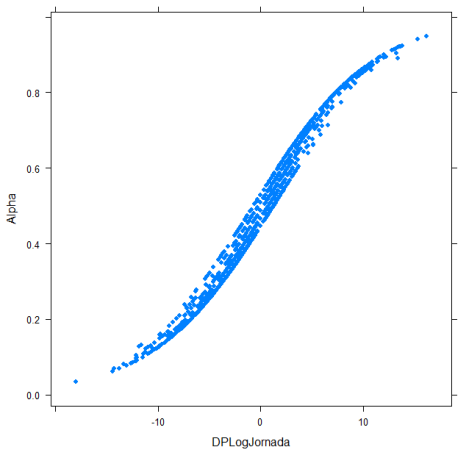


Figura 2.2: Relación α y *DPLogJornada*

1. Desarrollar un modelo de regresión logística paso a paso pero sin la variable *DPJornada* en la base de datos. *R* nos ofrece un modelo sólo con *DPLogJornada* como variable predictora.

2. Desarrollar otro modelo de regresión logistica paso a paso, pero esta vez quitando la variable *DPLogJornada* de la base de datos. En este caso *R* nos ofrece un modelo con *RV1* y *DPJornada* como variables predictoras.

Analizando los dos modelos nos hemos decantado por el primero, ya que en el segundo la variable predcitora *RV1* tiene un $\rho-$valor de $0,11$, muy elevado como para considerarla variable significativa. Un resumen del modelo final es el siguiente(ordenes en el anexo):

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.03578    0.06968  -14.87   <2e-16 ***
DPLogJornada -0.18659    0.01566  -11.92   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1550.0  on 1279  degrees of freedom
Residual deviance: 1372.3  on 1278  degrees of freedom
AIC: 1376.3

Number of Fisher Scoring iterations: 4
```

Es interesante ver que para predecir la victoria del equipo visitante sólo es necesario saber los puntos que tienen cada equipo en la jornada correspondiente y no importa ningún tipo de racha. Los estimadores de los parámetros son:

- $\hat{\beta}_0 = -1,036$
- $\hat{\beta}_2 = -0,187$

Por lo tanto, cada vez que aumentamos en una unidad la variable *DPLogJornada*, el *odds ratio* se incrementa en $e^{-0,187}$. Es muy interesante ver que el coeficiente $\hat{\beta}_2$ es prácticamente igual que en el modelo *victoria local* pero cambiado de signo. Esto es debido a que una diferencia de puntos positiva entre el local y el visitante influye positivamente a la hora de predecir la victoria local pero negativamente cuando se predice la victoria visitante.

Luego la probabilidad γ de que en un determinado partido, gane el equipo visitante, viene dada por:

$$\gamma = P(Y = 1|X_2 = x_2) = \frac{e^{-1,036-0,187x_2}}{1+e^{-1,036-0,187x_2}}$$

Realizando la trasnformación *logistica* para linealizar, obtenemos lo siguiente:

$$log\left(\frac{\gamma}{1-\gamma}\right) = -1,036 - 0,187x_2$$

De esta manera ya tenemos los dos modelos para predecir la probabilidad de que en un cierto partido de fútbol, gane el equipo local o gane el equipo visitante a partir de unas variables predictoras muy intuitivas y sencillas de obtener.

2.4. Validación de los modelos

Para comprobar si los resultados que dan nuestros modelos son aceptables vamos a realizar varias pruebas sobre los datos con los que los hemos creado (de la jornada 4 a la jornada 35 por las razones mencionadas anteriormente). En primer lugar, vamos a contabilizar las veces que da como suceso más probable la victoria local contrastándolo con los datos de dichas temporadas. Los resultados para el modelo *victoria local*, considerando que el modelo predice éxito cuando $\alpha > 0,5$ son los siguientes:

Predicción | Resultado	Victoria local	No victoria local
Victoria local	370(66,1 %)	190(33,1 %)
No victoria local	244(33,9 %)	476(66,1 %)

Cuadro 2.1: Tabla que representa la predicción del modelo *victoria local* con el resultado que se dio en cada partido con sus respectivos porcentajes por filas.

Los resultados son favorables al modelo ya que de 1280 resultados se aciertan 846, lo que supone un 66,1 %. Veamos también el mismo contraste pero con el modelo *victoria visitante* considerando que el modelo predice éxito cuando $\gamma > 0,5$.

Predicción | Resultado	Victoria visitante	No victoria visitante
Victoria visitante	93(59,6 %)	63(40,4 %)
No victoria visitante	283(25,2 %)	841(74,8 %)

Cuadro 2.2: Tabla que representa la predicción del modelo *victoria visitante* con el resultado que se dio en cada partido con sus respectivos porcentajes por filas.

En este caso los resultados también son favorables al modelo ya que acierta 934 de 320 resultados, lo que supone un 72,96 %. Mirando a una posible aplicación del modelo al mundo de las apuestas, cabe resaltar que no nos interesa tanto si el equipo va a ganar o no, si no que queremos estimar las probabilidades de forma que sean lo más cercanas a la realidad posible, para compararlas con las cuotas y decidir si merece la pena apostar. Por ejemplo, en algunas situaciones merece más la pena apostar a un equipo con probabilidad de victoria de 0,4 pero con cuota de 21€ , que a un equipo con probabilidad de victoria de 0,8 pero con cuota de 1,08€.

A simple vista los resultados parecen aceptables, para comprobar si finalmente lo son vamos a realizar lo que se conoce como curva ROC. La curva ROC es una herramienta estadística utilizada en el análisis de la bondad de modelos con variable respuesta de carácter dicotómico. La curva ROC

representa las medidas de sensibilidad y especificidad, por tanto veamos previamente qué son estos dos conceptos en el ámbito de partidos de fútbol (centrándonos sólo en la victoria local ya que es análogo para la victoria visitante).

- Llamamos **sensibilidad** a la probabilidad de, dado un partido en el que gana el local, el modelo prediga que gana el local. También esto se conoce como *razón de verdaderos positivos*.

- Llamamos **especificidad** a la probabilidad de, dado un partido en el que no gana el equipo local, el modelo prediga que no gana el equipo local. También esto se conoce como *razón de verdaderos negativos*.

Notar que tanto la sensibilidad como la especificidad depende de un parámetro α que es el punto de corte a partir del cuál se predice que el equipo local gana. Para calcularlas, es necesario tener en cuenta los siguientes conceptos:

- Denotamos V_+^α a los verdaderos positivos, es decir, a los partidos en los que gana el local y el modelo predice que gana el local (es decir, si $p \geqslant \alpha$).

- Denotamos V_-^α a los verdaderos negativo, es decir, a los partidos en los que pierde el local y el modelo predice que pierde el local (es decir, si $p < \alpha$).

- Llamamos falsos positivos F_+^α a los partidos en los que pierde el local y el modelo predice que gana ($p \geqslant \alpha$).

- Llamamos falsos negativos F_-^α a los partidos en los que gana el local y el modelo predice que no gana ($p < \alpha$).

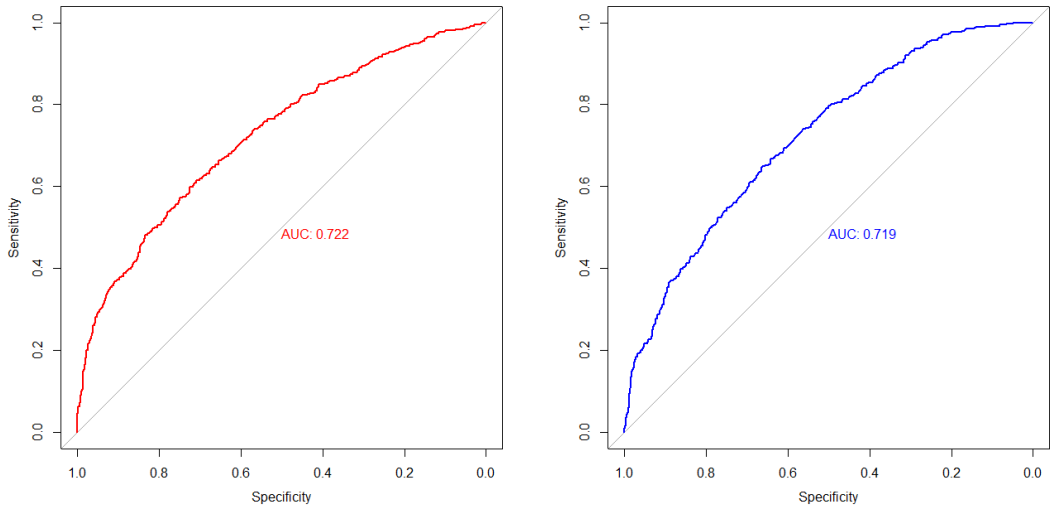
Con esta notación, la *sensibilidad* S^α es

$$S^\alpha = \frac{V_+^\alpha}{V_+^\alpha + F_-^\alpha}$$

y la *especificidad* E^α es

$$E^\alpha = \frac{V_-^\alpha}{V_-^\alpha + F_+^\alpha}$$

La curva ROC representa $1 - especificidad$ frente a la *sensibilidad* para cada posible valor α. Para comprobar la bondad de un modelo a partir de la curva ROC generada se utiliza habitualmente el área bajo la curva (AUC), y si se da que $AUC > 0,7$, entonces el modelo tiene un valor de diagnóstico aceptable. Las curvas ROC obtenidas de los modelos *victoria local* y *victoria visitante* son las siguientes:



(a) Victoria local (b) Victoria visitante

Figura 2.3: Curvas ROC de ambos modelos

Como vemos los valores del AUC están en torno a $0,7$ y por lo tanto los modelos se consideran adecuados. Veamos ahora las mismas comprobaciones pero sobre la temporada $2017-2018$. Análogamente, podemos validar los modelos aplicándolos a la temporada $2017-2018$; la comparativa de lo que predicen los modelos *victoria local* y *victoria visitante* con lo que sucede en realidad se puede comprobar en los Cuadros 2.3 y 2.4 respectivamente.

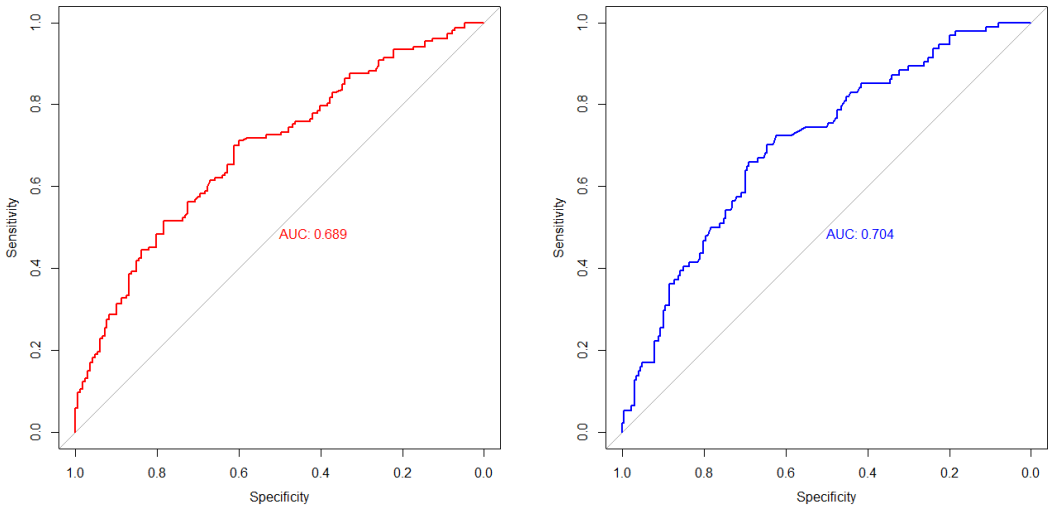
Predicción | Resultado	Victoria local	No victoria local
Victoria local	$89(63,1\,\%)$	$52(36,9\,\%)$
No victoria local	$64(35,8\,\%)$	$115(64,2\,\%)$

Cuadro 2.3: Tabla que representa la predicción del modelo *victoria local* con el resultado que se dio en cada partido con sus respectivos porcentajes por filas en la temporada $2017-2018$.

Predicción | Resultado	Victoria visitante	No victoria visitante
Victoria visitante	$20(52,6\,\%)$	$18(47,4\,\%)$
No victoria visitante	$74(26,2\,\%)$	$208(73,8\,\%)$

Cuadro 2.4: Tabla que representa la predicción del modelo *victoria visitante* con el resultado que se dio en cada partido con sus respectivos porcentajes por filas en la temporada $2017-2018$.

En esta temporada, los resultados también son favorables a los modelos, ya que el modelo *victoria local* acierta 204 de 320 lo que supone un $63,75\,\%$ y el modelo *victoria visitante* acierta 228 de 320 resultados lo que supone un $71,25\,\%$. Comprobamos también la bondad de los modelos en esta temporada mediante las curvas ROC:



(a) Victoria local (b) Victoria visitante

Figura 2.4: Curvas ROC de ambos modelos en la temporada 2017 – 2018

Los valores del AUC están ligeramente por debajo de los de la Figura 2.3, lo que es lógico ya que los datos de la temporada 2017 – 2018 no se han usado para el ajuste. En cualquier caso se encuentran muy cercanos a $0,7$ lo que puede interpretarse como aceptable.

Capítulo 3

Determinación de una estrategia óptima de apuesta

3.1. Introducción

Una vez obtenidos los modelos que predicen resultados de fútbol, el objetivo final es usarlos para obtener beneficio económico en las casas de apuestas. A diferencia de muchos otros modelos en los que se apuesta siempre una cantidad fija, permitiremos apostar una cantidad de dinero variable en cada partido para maximizar la ganancia. Para ello, hemos pensado en varias alternativas; en primer lugar repasaremos el clásico *Criterio de Kelly* y veremos si es adecuado para nuestros modelos, y posteriormente desarrollaremos un problema de *programación dinámica* que nos maximice la probabilidad de conseguir una cantidad de dinero en un cierto número de jornadas. Esta segunda opción parece a priori bastante adecuada dado que el desarrollo de un problema de programación dinámica se basa en una toma de decisiones secuencial, luego se ajusta bastante a la idea de ir apostando a varios partidos en cada una de las jornadas de la temporada.

3.2. Criterio de Kelly

El criterio de Kelly, como se indica en [7], es un procedimiento probabilístico cuyo objetivo es maximizar la esperanza del beneficio cuando se apuesta en un juego favorable. Por "juego favorable" entendemos aquel en el que existe una estrategia de apuesta tal que $P(\lim_{n\to\infty} X_n = +\infty) > 0$, donde X_n es el capital después de n repeticiones del juego. Veamos la idea del criterio de Kelly con un sencillo ejemplo:

Imaginemos un juego contra alguien infinitamente rico en el que se intenta adivinar cómo caerá una moneda después de un lanzamiento. Suponemos que la moneda tiene probabilidad de cara $p > 1/2$ y vamos a apostar siempre cara. En el caso de que ganemos, el beneficio obtenido será el apostado y en el caso de que perdamos, la pérdida es igual a la cantidad apostada. Al comienzo del juego, nuestro capital inicial es de X_0 y el principal problema se basa en elegir qué cantidad B_i apostamos en la i-ésima repetición. Un criterio clásico es el de elegir B_i para cada i de manera que el valor esperado $E(X_n)$ sea máximo después de n repeticiones. Tomando $T_k = 1$ si el k-ésimo resultado es cara y $T_k = -1$ si es cruz, entonces $X_k = X_{k-1} + T_k B_k$ para $k = 1,2,3,\ldots$ y por lo tanto se sigue que $X_n = X_0 + \sum_{k=1}^{n} T_k B_k$. Se tiene

$$E(X_n) = X_0 + \sum_{k=1}^{n} E(T_k B_k) = X_0 + \sum_{k=1}^{n} (p-q)E(B_k), \; con\; q = 1-p \tag{3.1}$$

Como $p - q > 0$, el juego tiene una esperanza positiva, luego en lugar de maximizar $E(X_n)$ vamos a maximizar $E(B_k)$ en cada repetición. Así pues, para maximizar la esperanza de la ganancia debemos apostar todo nuestro capital en cada repetición: inicialmente $B_1 = X_0$, si ganamos, $B_2 = 2X_0$ y así sucesivamente. Sin embargo, la probabilidad de arruinarnos apostando de esta manera viene dada por

$1-p^n$, y con $1/2 < p < 1$ se da que $\lim_{n\to\infty}(1-p^n) = 1$, por lo que la ruina es segura, así que apostar para maximizar la ganancia no es una buena estrategia.

Por otro lado, si apostamos con el objetivo de minimizar la probabilidad de ruina (entendemos ruina cuando $X_k = 0$ para algún k), llegamos a la conclusión de que tenemos que apostar lo mínimo posible en cada repetición. Sin embargo, ésta tampoco es una buena estrategia ya que estamos minimizando simultáneamente la esperanza de la ganancia media.

Una estrategia intermedia que esté entre maximizar $E(X_n)$ (asegurando la ruina) y minimizar la probabilidad de ruina (minimizando también $E(X_n)$) fue propuesta por J.L. Kelly.

En el juego de la moneda antes mencionado, donde la probabilidad de ganar y el beneficio son siempre iguales, parece intuitivo que la estrategia "óptima" es apostar siempre la misma fracción f de nuestra fortuna. Por consiguiente, si apostamos de forma que $B_i = fX_{i-1}$, donde $0 \leqslant f \leqslant 1$, estaremos siempre apostando el mismo porcentaje de nuestro capital (lo que se conoce como *fracción fija*). Es fácil ver que de esta manera, si denotamos por S al número de veces que ganamos el juego y por F al número de veces que perdemos, se sigue que $X_n = X_0(1+f)^S(1-f)^F$, donde $S+F=n$, y si además tomamos $0 < f < 1$, se tiene que $P(X_n = 0) = 0$, luego nunca se alcanzará el estado de ruina.

Dado que

$$e^{n\log[X_n/X_0]^{1/n}} = X_n/X_0$$

la cantidad

$$log\left[\frac{X_n}{X_0}\right]^{1/n} = \frac{S}{n}log(1+f)+\frac{F}{n}log(1-f)$$

mide el coeficiente exponencial del incremento de la fortuna por cada jugada. Kelly propone maximizar la función $G(f)$, donde

$$G(f) = E\left\{log\left[\frac{X_n}{X_0}\right]^{1/n}\right\} = E\left\{\frac{S}{n}log(1+f)+\frac{F}{n}log(1-f)\right\} = p\,log(1+f)+q\,log(1-f)$$

Por tanto

$$G'(f) = \frac{p}{1+f}-\frac{q}{1-f} = \frac{p-q-f}{(1+f)(1-f)} = 0$$

con $f = p-q$. Además como se tiene que

$$G''(f) = \frac{-f^2+2f(p-q)-1}{(1-f^2)^2} < 0$$

se sigue que $G'(f)$ es monótona y estrictamente decreciente en $[0,1)$. Además, dado que $G'(0) = p-q > 0$, $\lim_{f\to 1^-} G'(f) = -\infty$ y por continuidad de $G'(f)$, $G(f)$ tiene un único máximo en $f^* = p-q$, como se puede apreciar en la Figura 3.1. Es decir, el criterio de Kelly propone apostar la fracción $p-q = 2p-1$ en cada jugada. La explicación de por qué Kelly eligió maximizar la función $G(f)$ puede encontrarse detalladamente con demostraciones en [9].

El criterio de Kelly puede ser extendido con facilidad a juegos dónde no siempre tenemos la misma cuota. Sea $b = cuota - 1$ y suponiendo que en cada repetición la probabilidad de ganar es $p > 0$ y $bp - q > 0$, entonces el juego sigue siendo favorable y el procedimiento descrito anteriormente puede ser también usado para maximizar

$$G(f) = E\left\{log\left[\frac{X_n}{X_0}\right]\right\} = p\,log(1+bf)+q\,log(1-f)$$

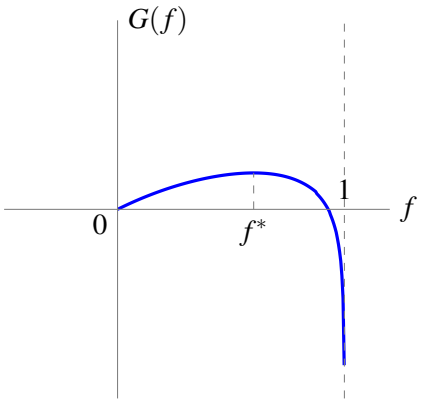


Figura 3.1: Gráfica de la función $G(f)$ en el intervalo $[0,1]$ con $p = 0,8$.

y con los mismos argumentos se llega a que el único máximo se alcanza en $f^* = (bp-q)/b$. Esto es, el criterio de Kelly sigue apostar la fracción $(bp-q)/b$ cuando el juego es favorable.

El gran problema que tiene el criterio de Kelly es que parte del supuesto de que el depósito inicial sea infinitamente divisible y esto en la práctica no se cumple.

Veamos que pasa si aplicamos el criterio de Kelly a nuestros modelos en la temporada $2017-2018$. Para ello hemos calculado la cantidad $E(beneficio)$ y, en el partido donde esta cantidad sea más alta (y positiva), hemos apostado la parte entera de la fracción $(bp-q)/b = (cp-1)/(c-1)$ de nuestro capital, siendo p y c la probabilidad de victoria y su cuota respectivamente. Empezando con 10€ en la *jornada* 4 la estrategia según el criterio de Kelly es (denotando por B a nuestro capital):

- En el partido S.D. Eibar - C.D. Leganés (1-0), la $E(beneficio)$ del S.D. Eibar es de $0,7$ y la del C.D. Leganés es de $1,38$, luego apostamos al C.D. Leganés ($p = 0,34$ y $c = 3,8$) la fracción $f = 0,134$ de nuestro capital, es decir, $\lfloor 10 \cdot 0,134 \rfloor = \lfloor 1,34 \rfloor = 1$€ y lo perdemos. Luego $B = 9$.

- En el partido Getafe C.F. - F.C. Barcelona (1-2) de la jornada 4, razonando de la misma manera, nos dice apostar al Getafe C.F. ($p = 0,32$ y $c = 11,6$) la fracción $f = 0,25$ de nuestro capital, luego le apostamos $\lfloor 9 \cdot 0,25 \rfloor = \lfloor 2,25 \rfloor = 2$€ y los perdemos, es decir, $B = 7$€.

- En el partido Levante U.D. - Valencia C.F. (1-1), nos dice apostar al Levante U.D. ($p = 0,46$ y $c = 3,15$) la fracción $f = 0,22$ de nuestro capital, es decir, 1€ y lo perdemos, por tanto $B = 6$.

- En el partido Real Sociedad - Real Madrid C.F. (1-3), apostamos a la Real Sociedad ($p = 0,610$ y $c = 4,75$ la fracción $f = 0,5$ de nuestro dinero, es decir, 3€ que los perdemos y por consiguiente $B = 3$

- En el partido Girona - Sevilla F.C. (0-1), nos dice apostar al Girona ($p = 0,36$ y $c = 3,25$) la fracción $f = 0,06$ de nuestro capital, es decir, $0,2$€ pero cómo la apuesta mínima de la casa de apuestas es de 1€ lo apostamos y lo perdemos, por tanto $B = 2$.

- En el partido C.D. Alavés - Villarreal C.F. (0-3), apostamos al C.D. Alavés ($p = 0,36$ y $c = 3,01$) la fracción $0,05$ de nuestro capital, y por el mismo razonamiento apostamos 1€ y lo perdemos, por lo tanto $B = 1$.

- Pasamos a la *jornada* 5 y nos aconseja apostar la fracción $f = 0,11$ al S.D. Eibar ($p = 0,15$ y $c = 21$ partido F.C. Barcelona - S.D. Eibar (6-1), es decir, apostamos 1€ y lo perdemos, por tanto $B = 0$.

En consecuencia, como hemos comprobado en la *jornada* 5 ya hemos alcanzado la ruina, luego criterio de Kelly no es conveniente para nuestro problema. De hecho, este criterio ha sido criticado por

varios autores, [11], y se recomienda como regla general, apostar menos de lo que dice el criterio para evitar una ruina temprana.

3.3. Programación dinámica

La programación dinámica proporciona un procedimiento para determinar la combinación de decisiones que maximiza el beneficio total o minimiza el costo total en problemas en los que ha de tomarse un conjunto de decisiones secuencialmente. La programación dinámica trata el problema, que puede contener un gran número de variables interrelacionadas, como si estuviera formado por una sucesión de problemas, cada uno de los cuales tuviera sólo unas pocas variables. Puede aplicarse a problemas cuya representación matemática inicial es bastante diferente y no existe un planteamiento matemático. Pueden ser deterministas o estocásticos (nuestro problema será de carácter estocástico).

Las características generales de un problema de programación dinámica son las siguientes:

- El problema puede ser dividido en *etapas*, de manera que se ha de tomar una decisión en cada etapa.
- Cada etapa tiene asociada un cierto número de *estados*. Llamamos *estado* a toda la información que se precisa en cualquier etapa para poder tomar una decisión óptima.
- La decisión elegida en cualquier etapa describe cómo se transforma el estado en la etapa actual en el estado en la etapa siguiente. En el caso de programación dinámica estocástica, el estado actual y la decisión tomada determinan la distribución de probabilidad del estado en la etapa siguiente.
- **Principio de optimalidad**. Dado el estado actual, la decisión óptima para cada una de las restantes etapas no depende de los estados en los que se ha estado previamente o de las decisiones previamente tomadas.

Resumiendo, el planteamiento de un problema de programación dinámica se basa en un sistema que evoluciona a lo largo de un cierto número de etapas, en cada una de las cuales, el sistema está descrito por un conjunto de parámetros denominados estados. En cada etapa, independientemente del estado en el que se encuentre el sistema, debe de tomarse una decisión, y para tomar esta decisión, no es relevante la historia pasada del sistema (lo que hemos mencionado como principio de optimalidad). Cuando se toma una decisión, se obtiene un pago (beneficio o costo) que depende del estado en el que estaba el sistema y de la decisión tomada. El propósito del problema es optimizar alguna función del estado inicial y de las decisiones subsiguientes. La notación empleada es la siguiente:

- N denota el número de etapas.
- $x_k,\ k = 0,\ldots,N-1$ denota el estado en la etapa k, que recoge toda la información que se precisa en cada etapa para tomar la decisión óptima. El estado al comienzo del problema es x_0.
- $d_k,\ k = 0,\ldots,N-1$ denota la variable decisión. Se ha de seleccionar en la etapa k sabiendo que el estado del sistema es x_k. El conjunto de decisiones posibles del sistema dependerá, en general, del estado del sistema. Se denota por $D_k(x_k)$.
- $w_k,\ k = 0,\ldots,N-1$ denota la distribución de probabilidad. Representa el parámetro aleatorio que afecta la transición entre estados y los pagos generados. Las variables w_k son independientes aunque su distribución puede depender de x_k y d_k.
- Ecuación de transferencia, $x_{k+1} = f_k(x_k, d_k, w_k),\ k = 0,\ldots,N-1$. Indica como se transforma el estado actual x_k por efecto de la decisión d_k y de la componente aleatoria w_k, en el estado en la etapa siguiente x_{k+1}.

- $g_k(x_k, d_k, w_k)$, $k = 0, \ldots, N-1$ denota el pago en una etapa. Indica el beneficio o costo que el sistema genera en una etapa por estar en el estado actual x_k , tomar la decisión d_k y por el efecto de w_k.
- $g_N(x_N)$ denota el pago terminal. Indica el pago al terminar el proceso.
- J_k, $k = 0, \ldots, N$ denota la función objetivo que queremos maximizar o minimizar.
- $J^\star(x_0)$ denota el valor óptimo de la función objetivo, que como es obvio, depende del estado inicial del sistema.
- $d_k^\star(x_k)$ denota la decisión óptima que ha de tomarse en cada etapa, $k = 0, \ldots, N-1$, para cada uno de los estados x_k en los que puede estar el sistema en dicha etapa.

El objetivo final de un problema de programación dinámica estocástico es maximizar (o minimizar) el valor esperado de la suma de los pagos generados en cada etapa y por finalizar el proceso. Es decir, maximizar o minimizar la cantidad:

$$\max_{\{d_0,\ldots,d_{N-1}\}} E_{w_0,\ldots,w_{N-1}} \left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, d_k, w_k) \right\}$$

La manera utilizada para resolver este problema, es el siguiente algoritmo:

- Paso 1: Hacer $k = N$ y $J_N(x_N) = g_N(x_N)$.
- Paso 2: Hacer $k = k-1$, y para cada estado, determinar

$$J_k(x_k) = \max_{d_k \in D(x_k)} E_{w_k} \{ g_k(x_k, d_k, w_k) + J_{k+1}[f_k(x_k, d_k, w_k)] \}$$

- Paso 3: si $k = 0$ detener el algoritmo, en otro caso ir al Paso 2.

Teorema 3.1. *En todo problema de programación dinámica se tiene que $J^\star(x_0) = J_0(x_0)$, es decir, el valor óptimo de la función objetivo se obtiene en el último paso del algoritmo.*

Una demostración extendida de esta teorema puede encontrarse en [4]. Por lo tanto, resolviendo el algoritmo anterior obtenemos la solución a un problema de programación dinámica.

3.4. Planteamiento del problema

Una vez explicados los conceptos básicos, la notación y el algoritmo empleado en la programación dinámica, vamos a utilizarlos en nuestro problema. Queremos resaltar que parece muy adecuada la elección de la programación dinámica para nuestro caso ya que queremos diseñar una estrategia óptima que nos diga a qué equipos apostar y qué cantidad en cada jornada, es decir, tenemos que tomar una serie de decisiones secuencialmente y es precisamente en lo que se basa este tipo de problemas. Como nuestros modelos sólo predicen resultados de la jornada 4 a la jornada 35, sólo apostaremos en estas jornadas.

El planteamiento principal del problema es el siguiente: Dada una cantidad inicial B_i queremos maximizar la probabilidad de acabar con una cierta cantidad final B_f al cabo de N jornadas apostando a un número determinado de partidos. En nuestro caso, vamos a tomar $B_i = 10$ € y $B_f = 13$ € pero en lugar de realizar un sólo problema para obtener B_f, que empiece en la jornada 4 y termine en la 35, vamos a dividirlo en subproblemas de 4 en 4 jornadas, con el objetivo de conseguir B_f en cada uno de ellos. Para lograrlo, tenemos la opción de apostar a la victoria local y a la visitante de 4 partidos diferentes en cada una de las jornadas. Cabe destacar que:

- Si conseguimos nuestro objetivo antes de finalizar las 4 jornadas correspondientes, empezamos un nuevo problema.

- Como tenemos dos modelos diferentes para predecir la victoria local y la victoria visitante, consideramos cada partido como si fueran dos independientes por si la estrategia óptima es apostar tanto al local como al visitante. Además, a la hora de pensar en los distintos casos posibles y calcular sus respectivas probabilidades, cuando no gana el local no tiene por que ganar el visitante y viceversa, por consiguiente es más cómodo interpretarlos cómo partidos diferentes (pensando en la posterior programación en *Java*)
- Utilizamos cantidades pequeñas de dinero para no incrementar el coste computacional, pero el programa que diseñaremos es perfectamente aplicable a cantidades más elevadas. Por el mismo razonamiento hemos elegido apostar en cada jornada a cuatro partidos elegidos al azar en lugar de todos los de la jornada.

Además:

- $N = 4$. Cada una de las etapas identifica a una jornada diferente.
- x_k denota el dinero disponible al comienzo de la etapa k. Es obvio que si en algún momento $x_k = 0$ terminamos el problema sin cumplir nuestro objetivo y si $x_k \geqslant 13$ dejamos de apostar por que ya se consigue lo que queremos.
- $x_0 = B_i = 10$
- d_k^i denota al dinero apostado en el partido i correspondiente a la etapa k, con $i = 1,2,\ldots,8$ y $k = 1,\ldots,4$.
- w_k^i denota el factor aleatorio que determina el resultado del partido i, con $i = 1,2,\ldots,8$ y $k = 1,\ldots,4$.
- c_k^i denota la cuota de victoria del partido i correspondiente a la etapa k, con $i = 1,2,\ldots,8$ y $k = 1,\ldots,4$. Las cuotas las hemos obtenido mediante una media aritmética de las principales casas de apuestas del mundo, tanto físicas cómo de carácter online.

Con esta notación, la ecuación de transporte viene dada por:

$$x_{k+1} = \sum_{i=1}^{8} f_k(x_k, d_k^i, w_k^i) = x_k + \sum_{i=1}^{8} \begin{cases} d_k^i c_k^i & \text{si } w_k^i \textit{es ganar} \\ -d_k^i & \text{si } w_k^i \textit{es perder} \end{cases}$$

Para poder resolver este tipo de problemas estocásticos en los que se quiere maximizar una probabilidad, es necesario definir los pagos generados en cada etapa cómo $g_k(x_k, d_k^i, w_k^i) = 0$ con $k = 1,2,3$ y $\forall i$, y el pago terminal cómo

$$g_4(x_4) = \begin{cases} 1 & \text{si } x_4 \geqslant 13 \\ 0 & \text{si } x_4 < 13 \end{cases} \tag{3.2}$$

Por otro lado, como el objetivo de un problema genérico de estas características es

$$\text{máx}\{P(X_N \geqslant C_f)\} = \text{máx}\, E[\xi]$$

considerando ξ una variable aleatoria tal que

$$\xi = \begin{cases} 1 & \text{si } X_N \geqslant C_f \\ 0 & \text{si } X_N < C_f \end{cases}$$

se tiene que

$$P(x_4 \geqslant 13) = 1\, P[x_4 \geqslant 13] + 0\, P[x_4 < 13] = E[\xi] = E[\sum_{k=1}^{3} g_k(x_k, d_k^i, w_k^i) + g_4(x_4)] = E[g_4(x_4)] \tag{3.3}$$

y de aquí, por el teorema mencionado se sigue que para resolver cada uno de los problemas hay que calcular

$$J^{\star}(x_0) = \max_{d_k \in D} E\left\{g_4(x_4)\right\}$$

para lo que es necesario ir recorriendo el algoritmo mencionado en la Sección 3.2. Notar que aunque normalmente en programación dinámica basta con una resolución del problema para que nos dé las soluciones de todas las etapas, en nuestro caso no es así, ya que tenemos que resolver tantos problemas de programación dinámica como etapas ($N = 4$); es decir, el primer problema lo resolvemos para $N = 4$, luego resolvemos un segundo problema con $N = 3$, luego $N = 2$ y finalmente $N = 1$. La razón de esto es que son necesarias las cuotas y las probabilidades de todas las jornadas, pero como es natural, sólo tenemos las probabilidades y las cuotas de la jornada en la que se está. Para solventar este problema, hemos programado la primera jornada con las probabilidades y las cuotas reales, pero para las demás hemos utilizado probabilidades y cuotas ficticias. Estos datos ficticios pueden ponerse a nuestra invención, pero para hacer el problema más real hemos utilizado cinco cuantiles de las probabilidades predichas por nuestro modelo *victoria local* de la temporada 2017-2018, ya que es de la única temporada de la que teníamos las cuotas. Para no tener problemas, evitamos el máximo y el mínimo, y de esta manera los cuantiles elegidos son:

- $p_1 = q_{0,05} = 0,183$ correspondiente al partido Málaga C.F. - Real Sociedad de la jornada 34 cuya cuota local es de $4,48$.
- $p_2 = q_{0,275} = 0,358$ correspondiente al partido S.D. Eibar - At. de Madrid de la jornada 19 cuya cuota local es de $3,9$.
- $p_3 = q_{0,50} = 0,472$ correspondiente al partido C.D. Alavés - U.D Las Palmas de la jornada 15 cuya cuota local es de $2,2$.
- $p_4 = q_{0,725} = 0,611$ correspondiente al partido At. de Madrid - Valencia C.F. de la jornada 22 cuya cuota local es de $1,49$
- $p_5 = q_{0,95} = 0,818$ correspondiente al partido Valencia C.F - R.C.D Espanyol de la jornada 31 cuya cuota local es de $1,456$.

Notar que son 5 partidos en lugar de 4 que tenemos en la primera etapa y que sólo introducimos las cuotas y las probabilidades de ganar el local en lugar de poner también las de visitante, pero por el hecho de ser partidos ficticios el programa no considera si son probabilidades y cuotas de ganar el local o el visitante, ni tampoco el hecho de que sean 5 partidos (podían ser más o menos).

Debido a que tenemos cada partido como si fueran dos independientes, la cantidad de combinaciones de resultados que pueden darse es en torno a 2^8, además también tenemos que calcular todas las posibles decisiones, por ejemplo, si $x_k = 10$, las maneras distintas de apostar a 8 partidos (en realidad 9, por que hay un noveno partido que es "no apostar"), son las combinaciones con repetición de 10€ a repartir tomados de 9 en 9, es decir, el número combinatorio 18 sobre 9 que es igual 48620. Por esta razón, hemos diseñado un programa en *Java* que haga esto por nosotros y cuyo código se puede encontrar explicado con detalle en el anexo.

3.4.1. Aspectos importantes sobre la programación en *Java*

Hemos realizado un programa en *Java* en el que a partir de unos datos que nos pide por teclado; nuestro capital inicial, el capital que se quiere conseguir, el número de etapas para conseguir dicho objetivo, el número de partidos (no es necesario duplicar cada uno de ellos ya que lo hace el programa automáticamente), las probabilidades de ganar el local y el visitante de cada uno de esos partidos (introduciremos las que dan nuestros modelos) y las cuotas gana local y gana visitante de cada uno de estos

partidos, el programa nos escribe por pantalla la estrategia óptima que habría que seguir para alcanzar el objetivo en cada una de las etapas y la probabilidad que tenemos de alcanzarlo. Destacar que el código del programa no está sacado de ninguna fuente y es de creación propia.

En relación a cuáles son las cantidades a apostar, el programa nos permitirá apostar de 1 en 1 en el intervalo $[0, x_k]$ siendo k la jornada actual y de 5 en 5 en el intervalo $[25, x_k]$ siempre y cuando $x_k < x_N$ en cuyo caso el programa nos dirá que la solución óptima es no apostar (como es obvio). Recordar que en nuestro caso $x_N = 13$ y por tanto nunca se da la segunda opción aunque esté programada.

El funcionamiento central del programa se basa en calcular la función J en cada una de las etapas de la siguiente manera:

- Para la etapa N hemos definido una función que hace exactamente lo explicado en (3.2).
- Para las etapas comprendidas en $[2, N-1]$ calcula la función J para cada uno de los estados posibles utilizando las probabilidades y cuotas ficticias de la siguiente manera: Fijado un estado, calcula todas las combinaciones de apuestas y para cada una de esas combinaciones, analiza todos los posibles resultados (mediante código binario) que pueden darse, obteniendo la ecuación de transporte correspondiente. Mediante la función J de la etapa anterior y la probabilidad de que ocurran justamente esos resultados, el programa se queda con la decisión que maximiza el valor esperado, que por lo visto en (3.3) coincide la posibilidad de alcanzar el objetivo.
- Para la primera etapa hace exactamente lo mismo pero con las cuotas y las probabilidades introducidas por teclado.

El tiempo de ejecución del programa en un ASUS F540S con CPU Intel Dual-Core es de 6 minutos y 33 segundos.

3.5. Aplicación del programa y resultados obtenidos

Cómo ya tenemos unos modelos que nos predicen probabilidades y un programa que nos asegura una estrategia óptima para conseguir beneficios, lo lógico es aplicarlo a la vida real. Para ello vamos a ir ejecutando el programa diseñado en *Java* jornada a jornada de la temporada 2017-2018 y observar cuáles son los beneficios obtenidos. Es importante notar que en la aplicación del programa no se ha usado conocimiento previo de los resultados de la temporada $2017-2018$, y por tanto puede considerarse una simulación real, en el sentido de que se podrían haber hecho estas apuestas a lo largo de la temporada (es cierto que hemos usado los partidos de la temporada para obtener los cuantiles de las cuotas ya que son las únicas que teníamos, sin embargo para la temporada $2018-2019$ pueden usarse los mismos cuantiles y aplicar este procedimiento sin ningún conocimiento adicional de la temporada). Denotamos nuestro capital desde el comienzo de la simulación con B, y vamos a llamar *fases* a cada uno de los subproblemas de 4 jornadas.

Fase 1:

- Comenzamos en la *jornada* 4 con $x_0 = 10$, y el objetivo $x_N = 13$ en las próximas 4 jornadas (incluyendo ésta). Elegimos cuatro partidos al azar de la jornada, calculamos las probabilidades gana local y gana visitante con nuestros modelos y los introducimos en el programa con sus respectivas cuotas. Una vez ejecutado, el programa nos dice que la decisión óptima es apostar 1€ a la victoria del Getafe C.F. ($p = 0,319$, $c = 11,6$) en el partido Getafe C.F. - F.C. Barcelona (1-2), luego perdemos lo apostado y así $B = 9$. En esta jornada, la probabilidad de alcanzar el objetivo es de $0,88$.
- Pasamos a la *jornada* 5, pero como en la anterior hemos perdido dinero ahora empezamos con $x_0 = 9$ con el objetivo de conseguir $x_N = 13$ en las próximas 2 jornadas. El programa nos da como decisión óptima no apostar a nada, de manera que pasamos a la jornada siguiente. Ahora, la probabilidad de alcanzar el objetivo es de $0,82$.

- En la *jornada* 6 con $x_0 = 9$ y $x_N = 13$, nos da como solución óptima con probabilidad de alcanzar el objetivo de $0,65$, apostar 6€ a la victoria del R.C. Celta de Vigo ($p_1 = 0,224$, $c_1 = 3,25$) en el partido S.D Eibar - R.C. Celta de Vigo (0-2) y 3€ a la victoria del Sevilla C.F. ($p_2 = 0,304$, $c_2 = 5,875$) en el partido At. de Madrid - Sevilla C.F (2-0). Así pues, aunque pierde el Sevilla C.F. gana el Celta, luego se tiene que $B = 6 \cdot 3,25 = 19,5$ y cómo ya hemos llegado al objetivo, terminamos la fase.

Veamos ahora la **Fase 2**:

- Comenzamos la nueva fase empezando en la *jornada* 7 con $x_0 = 10$ y buscando el mismo objetivo que antes en 4 jornadas. Procediendo de la misma manera, el programa nos da cómo decisión óptima apostar 1€ a la victoria del C.D. Leganés ($p = 0,38$, $c = 6,5$) en el partido C.D Leganés - At. de Madrid (0-0), luego perdemos y $B = 9$.

- Pasamos a la *jornada* 8 con $x_0 = 9$ con el mismo objetivo pero con una jornada menos para apostar. La decisión óptima es no apostar a nada.

- En la *jornada* 9 da como solución óptima apostar 9€ a la victoria del Betis ($p = 0,676$, $c = 1,57$) en el partido Betis - Alavés $(2-0)$, luego $B = 9 \cdot 1,57 = 14,3$ y cómo se cumple el objetivo terminamos la fase.

Hasta ahora parece que el programa se comporta igual en todas las fases; apuesta 1€ en la primera jornada a un equipo con poca probabilidad de ganar pero con una cuota alta y lo pierde, la segunda jornada no apuesta y en la tercera lo hace de manera que consigue el objetivo. No vamos a detallar aquí jornada a jornada qué nos ofrece como estrategia óptima, pero hay un gran número de fases en los que la estrategia es diferente. Por ejemplo repasemos la **fase 7**. Empezamos en la *jornada* 22 con el capital inicial y objetivo de siempre. La decisión óptima dada es que apostemos 2€ a la victoria del Girona ($p = 0,522$, $c = 2,5$) en el partido Girona - Athletic Club (2-0), luego $B = 13$ y por lo tanto finalizamos la fase. También hay fases en las que la decisión óptima es distribuir bastante más nuestro dinero, algo muy interesante ya que no es una idea que se le puede ocurrir a cualquiera con facilidad. Sucede algo así en la **fase 13**:

- Empezamos en la *jornada* 32, en la que nos dice apostar 4€ al At. de Madrid ($p_1 = 0,871$, $c_1 = 1,25$) en el partido At. de Madrid - Levante U.D. (3-0) y 2€ a la victoria del Athletic Club ($p_2 = 0,679$, $c_2 = 1,6$) en el partido Athletic Club - R.C. Deportivo (2-3), luego ganamos la primera apuesta y perdemos la segunda, $B = 4 + 4 \cdot 1,25 = 9$.

- En la *jornada* 33 nos dice no apostar a nada.

- En la *jornada* 34 da como estrategia óptima apostar 5€ al C.D. Alavés ($p_1 = 0,466$, $c_1 = 2,29$) en el partido U.D. Las Palmas - C.D. Alavés (0-4). Por consiguiente $B = 2 + 5 \cdot 2,29 = 13,45$ y finalizamos la fase.

En la Figura 3.2 se puede apreciar cómo va evolucionando nuestro capital jornada a jornada de la temporada. Al finalizar la temporada, nuestro capital es de $58,96$€ de tal manera que el beneficio es de $48,96$€ por tanto los resultados son excelentes ya que sixtiplicamos nuestro capital inicial.

Como vemos, no en todas las jornadas se gana, y hay un momento de la temporada, en torno a la jornada 20 en la que se encadena una mala racha. Esto sucede en la **fase 6**, en la que:

- En la *jornada* 19 nos dice apostar 2€ al Valencia C.F ($p = 0,573$, $c = 2,28$) en el partido R.C. Deportivo - Valencia C.F. (1-2) así que ganamos y $B = 8 + 2 \cdot 2,28 = 12,56$.

- En la *jornada* 20 perdemos dinero por que nos aconseja apostar 3€ al At. de Madrid ($p = 0,738$, $c = 1,32$) en el partido At. de Madrid - Girona (1-1) y así $B = 9,56$.

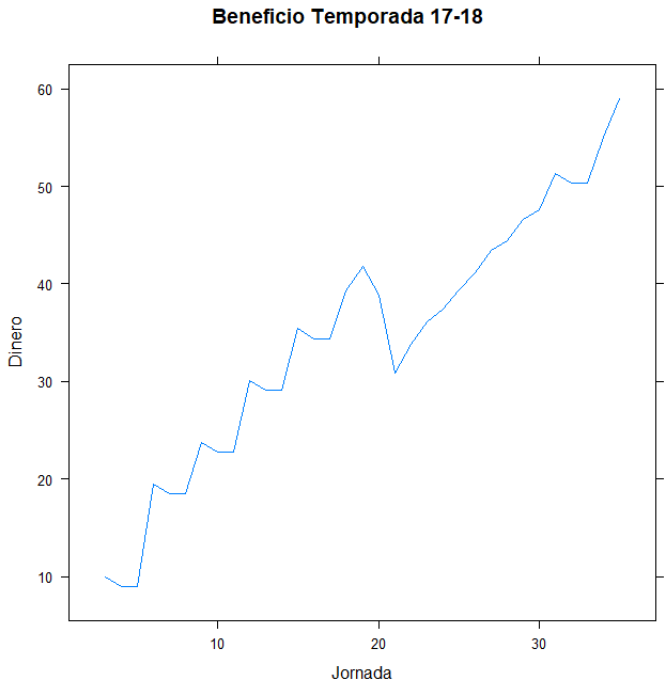


Figura 3.2: Beneficio obtenido durante la temporada 2017-2018 empezando con 10€

- En la jornada 21 volvemos a perder dinero ya que nos aconseja apostar 3€ al Levante ($p_1 = 0,286$, $c_1 = 4$) en el partido R.C Deportivo - Levante U.D. (2-2), 4€ al Girona ($p_2 = 0,471$, $c_2 = 3,05$) en el partido Málaga C.F - Girona (0-0) y 1€ a la U.D. Las Palmas ($p_3 = 0,056$, $c_3 = 17,08$) en el partido At. de Madrid - U.D. Las Palmas (3-0). Por tanto perdemos en las tres apuestas y así $B = 1,56$.

En total, obtenemos una pérdida de casi 9€ .

Este es el principal problema de optar por la programación dinámica, ya que aunque nos ponemos en cada fase un objetivo bastante alcanzable, arriesga todo lo que tiene para conseguirlo y así la etapa que pierde, pierde casi todo. Aunque los resultados son muy buenos, todavía se puede conseguir más beneficio; por ejemplo, si fijamos como objetivo $x_N = 14$ en cada fase con $x_0 = 10$, obtenemos un beneficio de $59,69$€ , por lo tanto incrementamos nuestra fortuna en un $600\,\%$. Una comparativa entre el beneficio obtenido con $x_N = 13$ y $x_N = 14$ puede verse en la Figura 3.3. Cabe destacar que no por aumentar el objetivo en todas las fases se consigue más beneficio, ya que para conseguirlo la estrategia a seguir es más arriesgada y se puede perder varias veces todo el dinero.

3.5.1. Alternativas y mejoras

Podríamos haber planteado algunos conceptos del problema de programación dinámica de manera alternativa. Algunos son las siguientes:

- Para las cuotas y probabilidades ficticias, en lugar de tomar los cuantiles de las probabilidades predichas por nuestros modelos, podemos usar las mismas que introducimos por teclado para todas las jornadas de la fase.
- Dado que la programación dinámica nos da en cada etapa la probabilidad de conseguir el objetivo, una manera de mejorar y obtener posiblemente mejor beneficio es poner un objetivo variable en cada una de las fases según las probabilidades que obtenemos por pantalla cuando ejecutamos el programa. Es decir, buscar el máximo x_N tal que la probabilidad de obtenerlo sea mayor que una fijada.

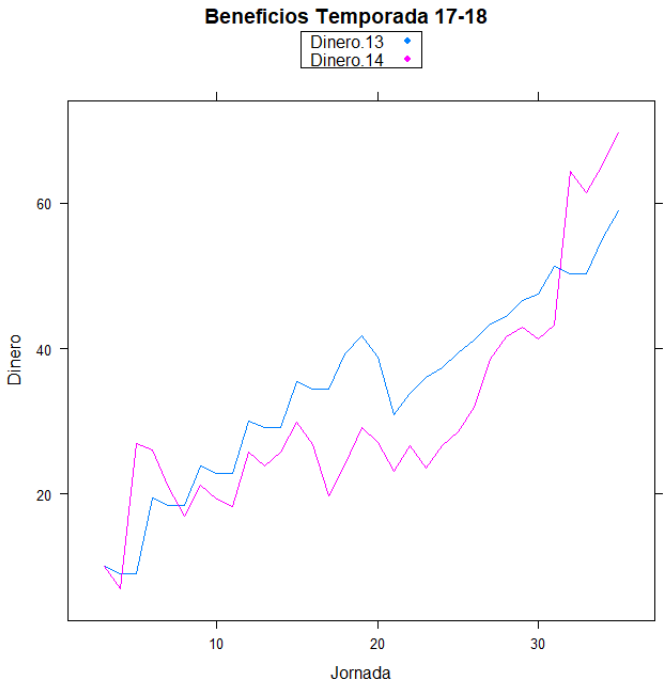


Figura 3.3: Beneficio obtenido durante la temporada 2017-2018 empezando con 10€ con diferentes objetivos por cada fase. En azul $x_N = 13$ y en rosa $x_N = 14$.

3.6. Conclusiones finales

- Podemos utilizar modelos estadísticos para predecir los resultados de fútbol de cualquier competición sólo viendo cómo están jugando en los últimos partidos y mirando su posición en la tabla. Aunque nuestros modelos no son mejores respecto a las predicciones de las casas de apuestas, son suficientemente buenos para obtener beneficio económico, como hemos comprobado.
- La programación dinámica se adapta a la perfección a un problema real de apuestas de partidos de fútbol, dando altos beneficios.
- El problema de intentar predecir un resultado de fútbol y el problema de programación dinámica para diseñar la estrategia óptima de apuesta son totalmente independientes. Hay modelos de predicción ya estudiados en la literatura que dan resultados más cercanos a la realidad que los nuestros y que se pueden usar con el programa de programación dinámica que hemos diseñado.

Apéndice A

Código R

```
BDLigaEsp <-
  readXL("C:/Users/sabro/Desktop/Carrera/TFG/Base de datos 2013-2018.xlsx",
  rownames=FALSE, header=TRUE, na="", sheet="Hoja3", stringsAsFactors=TRUE)

% MODELO PARA PREDECIR VICTORIA LOCAL

BDLigaEsp <- subset(BDLigaEsp, subset=Jornada>3)

BDLigaEsp <- subset(BDLigaEsp, subset=Jornada<36)

BDLigaEsp$DPJornada <- with(BDLigaEsp, DP/ Jornada)

BDLigaEsp$DPLogJornada <- with(BDLigaEsp, DP/ log(Jornada))


modelo1 <- glm(VL ~ RL1 + RV1 + RL2 + RV2 + DPJornada + DPLogJornada,
data = BDLigaEsp, family = "binomial")


modeloAIC <- stepwise(modelo1, direction = 'backward/forward' , criterion = 'AIC')


%Para la curva ROC

BDLigaEsp<- within(BDLigaEsp, {
  fitted.modeloAIC <- fitted(modeloAIC)
})

names(BDLigaEsp)[c(19)] <- c("PrediccionVL")

library(pROC)

rocVL<- roc(BDLigaEsp$VL, BDLigaEsp$PrediccionVL)
auc(rocVL)
plot(rocVL, col="red", print.auc=TRUE)

%Area under the curve: 0.722
```

```
%MODELO PARA PREDECIR VICTORIA VISITANTE


modelo2 <- glm(VV ~ RL1 + RV1 + RL2 + RV2 + DPJornada + DPLogJornada,
data = BDLigaEsp, family = "binomial")


modeloAIC2 <- stepwise(modelo2, direction = 'backward/forward' , criterion = 'AIC')

modelo3 <- glm(VV ~ RL1 + RV1 + RL2 + RV2 + DPLogJornada,
data = BDLigaEsp, family = "binomial")

modeloAIC3 <- stepwise(modelo3, direction = 'backward/forward' , criterion = 'AIC')


%Para la curva ROC

BDLigaEsp<- within(BDLigaEsp, {
  fitted.modeloAIC3 <- fitted(modeloAIC3)
})

names(BDLigaEsp)[c(20)] <- c("PrediccionVV")

rocVV<- roc(BDLigaEsp$VV, BDLigaEsp$PrediccionVV)
auc(rocVV)
plot(rocVV, col="red", print.auc=TRUE)

%Área bajo la curva: 0.7194
```

Apéndice B

Código Java

```
package tfg;

import java.util.Arrays;
import java.util.Scanner;

public class TFG {

    //probgen es un vector de reales que representa la probabilidad de ganar el
    //equipo local en el partido i pero obtenidos de forma genérica
    //Obtenidos por los cuaniles de mis predicciones.
    static double[] probgen = {0.1830564, 0.3576748, 0.4719785,
        0.6114536, 0.8183193};

    //cuotgen es un vector de reales que representa la cuota que ofrece la casa
    //de apuestas por que gane el local en el partido i de forma genérica
    //También son obtenidos a partir de la función de distribución.
    static double[] cuotgen = {4.48, 3.9, 2.2, 1.49, 1.456};

    //Probprim es un vector de reales que representa la probabilidad de victoria
    // de cada equipo en el partido i en la jornada en la que estamos.
    static double[] probprim;

    //cuotprim es un vector de reales que representa la cuota por ganar
    // de cada equipo en el partido i en la jornada en la que estamos.
    static double[] cuotprim;

    // ci es un entero que representa la cantidad de dinero inicial.
    static int ci;

    //dec es un real que representa la cantidad decimal
    //del dinero inicial.
    static double dec;

    //cf es un entero que representa la cantidad de dinero al
    //qué se quiere llegar.
    static int cf;

    // m es un entero que representa el número de partidos.
```

```
static int m;

// n es  un entero que representa el número de etapas.
//(En este caso, de jornadas)
static int n;

// matriz es una matriz que contiene en cada fila las probabilidades
// de ganar y de perder de un equipo equipo de la jornada en la que estamos.
static double[][] matriz;

//matrizgen es cómo la anterior pero para los partidos ficticios.
static double[][] matrizgen;

public static void main(String[] args) {

    //INTRODUCCION DE DATOS POR TECLADO
    Scanner sc = new Scanner(System.in);
    System.out.println("Introduce el depósito
    inicial (parte entera): ");
    ci = sc.nextInt();
    System.out.println("La parte decimal de la cantidad inicial: ");
    dec = sc.nextDouble();
    System.out.println("Introduce el depósito que se quiere obtener: ");
    cf = sc.nextInt();
    System.out.println("Introduce el número de partidos: ");
    m = sc.nextInt();
    System.out.println("Introudce el número de etapas: ");
    n = sc.nextInt();

    //GENERACIÓN DE MATRICES DE PROBABILIDADES (Útiles para cálculos)
    matriz = new double[2 * m][2];
    //Para poder apostar al local y al visitante
    for (int j = 0; j < m; j++) {
        System.out.println("Introduce la probabilidad de ganar el "
                + "local del partido " + (j + 1)
                + " de esta jornada: (con coma)");
        matriz[j][0] = sc.nextDouble();
        matriz[j][1] = 1.00000 - matriz[j][0];

    }

    for (int j = 0; j < m; j++) {
        System.out.println("Introduce la probabilidad de "
                + "ganar el visitante del partido " + (j + 1)
                + " de esta jornada: (con coma)");
        matriz[j + m][0] = sc.nextDouble();
        matriz[j + m][1] = 1.0 - matriz[j + m][0];

    }

    matrizgen = new double[5][2];
```

```
        for (int j = 0; j < matrizgen.length; j++) {
            matrizgen[j][0] = probgen[j];
            matrizgen[j][1] = 1.0 - matrizgen[j][0];
        }

        cuotprim = new double[2 * m];
        for (int j = 0; j < m; j++) {
            System.out.println("Introduce la cuota gana "
                    + "local del partido " + (j + 1)
                    + " de esta jornada: (con coma)");
            cuotprim[j] = sc.nextDouble();
        }

        for (int j = 0; j < m; j++) {
            System.out.println("Introduce la cuota "
                    + "gana visitante del partido " + (j + 1)
                    + " de esta jornada: (con coma)");
            cuotprim[j + m] = sc.nextDouble();
        }

        System.out.println();

        //LLAMADA AL PROGRAMA
        dinamica(n);

    }

    //FUNCIONES AUXILIARES
    //La función dinámica a partir del número de etapas, se encarga de llamar
    //a cada una de las funciones que me resuelven cada una de las etapas,
    // devolviendo la probabilidad que hay de alcanzar el objetivo en cada una
    // de ellas.
    public static double dinamica(int n) {
        System.out.println("------------Etapa " + n + " ------------");
        double[] j = ultimaEtapa(cf);
        System.out.println(Arrays.toString(j));
        for (int i = n - 1; i > 1; i--) {
            System.out.println("------------Etapa " + i + " ------------");
            j = etapa(j, cuotgen, matrizgen, 0.0);
            System.out.println(Arrays.toString(etapa(j, cuotgen, matrizgen, 0.0)));
        }
        System.out.println("------------Etapa " + 1 + " ------------");
        double res = primeraEtapa(j);
        System.out.println("La máxima probabilidad de acabar con "
                + "" + cf + " es " + res);

        return res;
    }

    //La funcion etapa, a partir de un vector de double J que representa
    // los valores que puede tomar dicha función, otro vector de
```

```
    // double c que representa a las cuotas, y una matriz mp
    // que representa las probabilidades, me devuelve
    // el vector J al hacer una etapa en un problema de programación dinámica.
    public static double[] etapa(double[] j, double[] c, double[][] mp, double decimal) {
        double[] jnext = new double[j.length];
        for (int h = 0; h < jnext.length; h++) {
            jnext[h] = j[h];
        }
        int[] x = estados(cf);
        int ind = 0;
        int[] vaux2 = new int[mp.length];
        for (int i = 0; i < x.length; i++) {
            double res = 0;
            int[] decisiones = auxiliar(x[i]);
            int max = maximo(decisiones);
            CombRep cr = new CombRep(decisiones.length, mp.length);
            int[] vaux = new int[mp.length];

            do {

                for (int h = 0; h < vaux.length; h++) {
                    vaux[h] = decisiones[cr.getPos(h)];
                }
                if (suma(vaux) <= max) {
                    double aux2 = 0;
                    int tam = (int) Math.pow(2, mp.length);
                    for (int k = 0; k < tam; k++) {
                        int b[] = binarioN(k, mp.length);
                        int z = transporte(x[i], vaux, b, c, mp.length, decimal);
                        aux2 = aux2 + jj(j, z) * probs(b, mp);
                    }
                    if (aux2 > res) {
                        res = aux2;
                        for (int q = 0; q < vaux2.length; q++) {
                            vaux2[q] = vaux[q];
                        }

                    }
                }
            } while (cr.next());
            System.out.println("La decisión tomada para " + ind + " es: ");
            System.out.println(Arrays.toString(vaux2));
            jnext[ind] = res;

            ind++;
        }
        return jnext;

    }

//  La función auxiliar me genera un vector de enteros que serán las posibles
```

```
//  cantidades a apostar dado un capital n.
    public static int[] auxiliar(int n) {
        if (n <= 25) {
            int[] aux = new int[n + 1];
            for (int i = 0; i < aux.length; i++) {
                aux[i] = i;
            }

            return aux;
        } else {
            int[] aux = new int[26 + ((n - 25) / 5)];
            for (int i = 0; i < 26; i++) {
                aux[i] = i;
            }
            for (int i = 26; i < aux.length; i++) {
                aux[i] = aux[i - 1] + 5;
            }
            System.out.println((int) ((n - 25) / 5));
            return aux;
        }
    }

// La función primeraEtapa hace lo mismo que la función etapa pero con las
// probabilidades y cuotas introducidas por teclado.
    public static double primeraEtapa(double[] j) {
        double jnext[] = etapa(j, cuotprim, matriz, dec);
        System.out.println(Arrays.toString(jnext));
        return jnext[ci];
    }

//  La función ultimaEtapa me realiza la última etapa de un problema de
//  programación dinámica en los que se maximiza probabilidades. Si el estado
// x_{N} es mayor o igual que el objetivo devuelve 1 y 0 en otro caso.
    public static double[] ultimaEtapa(int cf) {
        double[] j = new double[cf + 1];
        for (int i = 0; i < j.length; i++) {
            if (i < cf) {
                j[i] = 0;
            } else {
                j[i] = 1;
            }
        }

        return j;
    }

//  La función transporte representa la ecuación de transporte tal y como la
// hemos definido en el planteamiento del problema
    public static int transporte(int x, int[] d, int[] p,
            double[] c, int numpart, double decimal) {
        double aux = x + decimal;
```

```
        for (int i = 0; i < numpart; i++) {
            if (p[i] == 0) {
                aux = (aux - d[i]) + d[i] * c[i];
            }
            if (p[i] == 1) {
                aux = aux - d[i];
            }
        }

        return (int) aux;
    }

//  La función jj me representa la función de la función J de un problema de
// programación dinámica incluyendo el hecho de que deje de apostar cuando se
// alcanza el objetivo.
    public static double jj(double[] j, int i) {
        if (i >= j.length) {
            return 1.0;
        }
        if (i <= 0) {
            return 0;
        }
        return j[i];
    }

//  La función binarioN me devuelve un array que es la representación en binario
//  en n bits de un numero que le pasamos como argumento.
    public static int[] binarioN(int numero, int n) {
        StringBuilder ala = new StringBuilder();
        String numerobinario = "";

        numerobinario = numerobinario + (numero % 2) + " ";

        numero = numero / 2;

        while (numero >= 2) {

            numerobinario = numerobinario + (numero % 2) + " ";
            numero = numero / 2;
        }
        numerobinario = numerobinario + numero;

        StringBuilder cadena = ala.append(numerobinario);
        cadena = ala.reverse();
        String[] aux = cadena.toString().split(" ");
        int[] res = new int[n];
        int vax1 = res.length - aux.length;
        for (int i = 0; i < vax1; i++) {
            res[i] = 0;
        }
        for (int i = vax1; i < res.length; i++) {
```

```
            res[i] = Integer.parseInt(aux[i - vax1]);
        }

        return res;
    }

//  La función máximo me devuelve el máximo de un vector de enteros.
    public static int maximo(int[] v) {
        int max = 0;
        for (int i = 0; i < v.length; i++) {
            if (max < v[i]) {
                max = v[i];
            }
        }
        return max;
    }

//  La función suma me devuelve la suma de todas las componentes de un vector.
    public static int suma(int[] v) {
        int res = 0;
        for (int i = 0; i < v.length; i++) {
            res = res + v[i];
        }
        return res;
    }
//  La funcion estados dado un entero cf, genera el array [0, 1, ..., cf]

    public static int[] estados(int cf) {
        int[] x = new int[cf + 1];
        for (int i = 0; i < x.length; i++) {
            x[i] = i;
        }
        return x;
    }

//  La función probs me devuelve la probabilidad de que se de un suceso, dada
//  una combinación de resultados y una matriz con las probabilidades de cada
//  uno de esos resultados.
    public static double probs(int[] bin, double[][] mp) {
        double res = 1;
        for (int i = 0; i < mp.length; i++) {
            if (bin[i] == 0) {
                res = res * mp[i][0];
            }
            if (bin[i] == 1) {
                res = res * mp[i][1];
            }
        }

        return res;
    }
```

```
}


//La clase CombRep genera a partir de un vector de posibles cantidades apostar,
//todas las posibles maneras de repartir nuestro dinero a cada partido.
//Para ello es necesario hacerlo con un vector de indices que posteriormente
// se redirigirá
public class CombRep {

    private int[] comb;
    private int limite;

    public CombRep(int n, int m) {
        comb = new int[m];
        this.limite = n - 1;
    }

    public boolean next() {
        int i = comb.length - 1;
        while (i >= 0 && comb[i] == limite) {
            i--;
        }
        if (i < 0) {
            return false;
        }
        comb[i]++;
        for (int j = i + 1; j < comb.length; j++) {
            comb[j] = 0;
        }
        return true;
    }

    public int getPos(int i) {
        return comb[i];
    }

}
```

Bibliografía

[1] S.CHATTERJEE y A.S. HADI, *Regression Analysis by Example*, John Wiley & Sons, Inc, 2006.

[2] S.J. SHEATHER, *A Modern Approach to Regression with R*, Springer Science+Business Media LLC, 2009.

[3] D.P. BERTSEKAS, *Dynamic Programming. Deterministic and Stochastic Models*, Prentice-Hall Inc., Englewood Cliffs, NJ, 1987.

[4] S.E. DREYFUS y A.M. LAW, *The Art and Theory of Dynamic Programming*, Academic Press, San Diego, CA, 1977.

[5] B.D SIVAZLIAN y L.E. STANFEL, *Optimization Techniques in Operations Research*, Prentice-Hall Inc., Englewood Cliffs, NJ, 1975.

[6] W.L WINSTON, *Operations Research*, Thomsom Brooks/Cole, Belmont, CA, 4th edición, 2004.

[7] W. FELLER, *An introduction to Probability Theory and Its Applications*, Vol. I, Revised(1966). New York, John Wiley.

[8] L.M. ROTANDO y E.O. THORP, *The Kelly Criterion and the Stock Market*, `http://www.jstor.org/stable/2324484`.

[9] E.O. THORP, *Optimal gambling systems for favorable games*, Review of the International Statistical Institute, Vol. 37:3, 1969.

[10] L. BREIMAN, *Optimal gambling systems for favorable games*, Fourth Berkeley Symposium on Probability and Statistics, 1961.

[11] L.C. MACLEAN, E.O. THORP y W.T. ZIEMBA, *Good and bad properties of the Kelly criterion*, Dalhousie University, Halifax, NS, 2010.

[12] DIARIO SPORT, *La final del mundial de 2014 sigue siendo la más vista*, `https://www.sport.es/es/noticias/mundial-futbol/final-del-mundial-sigue-siendo-mas-vista-4885285`.

[13] ACTUALIDAD VALDEPEÑAS, *El boom de las apuestas deportivas en España en 2017*, `https://www.actualidadvaldepenas.com/articulo/ocio/boom-apuestas-deportivas-espana-2017/20180201075322124236.html`.