Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

Análisis básico y avanzado de datos: Posiciones de fútbol

Fundamentos de Ingeniería de Datos

Autores:
José Luis Alonso Rocha
Fernando Alonso Martín
Félix Jiménez Gonzalez
Vincenzo Parretta

Enero 2023

Índice

1. Planteamiento y estructura general del análisis

El trabajo de análisis de datos a abarcar en este proyecto de data mining se centra en la exploración y construcción de diversos modelos a través de diversas pruebas y parámetros para encontrar el modelo óptimo con el que obtener los mejores resultados. En nuestro caso, la temática adaptada gira en torno a fútbol, centrando el problema a tratar en el análisis de un extenso conjunto de datos para predecir posiciones generales de jugadores de fútbol, basándose en estadísticas extraídas de su rendimiento en los partidos jugados.

El objetivo de este documento es explicar de forma general el desarrollo del trabajo, así como los resultados obtenidos, de forma que pueda servir como informe de resultados del proyecto, así como la justificación del trabajo realizado por cada una de las parejas, tal como se indica en los criterios del proyecto de la asignatura.

En los planteamientos iniciales del grupo para como abarcar el proyecto, analizando posibles ideas con las que explotar la rica fuente de datos extraída, se llegó a la siguiente conclusión, a través de la imposición de las siguientes tareas:

- Para cubrir el criterio de **análisis de datos básico** se determinó el desarrollo de una tarea de clasificación, respecto al conjunto de datos extraído. Se haría uso de las etiquetas y campos ya disponibles en el propio archivo a tratar para poder llevar a cabo el desarrollo y análisis de la idea. La tarea es simple: predecir individualmente las posiciones de los jugadores de fútbol a través de sus estadísticas basándonos en su rendimiento en el año.
  Esta parte del trabajo, respectiva al aprendizaje supervisado, ha sido desarrollada por **José Luis Alonso Rocha** y **Fernando Alonso Martín**

- En siguiente lugar, para cubrir el criterio de **análisis de datos avanzado**, se desarrollaría una tarea de clustering. La idea inicial partiría de la base de que cada clúster sería las posibles posiciones de los jugadores, definido a través de cada uno de los puntos (que serían los jugadores) que se agruparían en torno a los distintos clústers que se formen. De esta forma se pueden llegar a concluir diversos resultados en torno a la distancia entre los clústers, como por ejemplo, la similitud que hay entre un jugador en la posición defensa con los mediocentro-defensa.
  Esta parte del trabajo, respectiva al aprendizaje no supervisado, ha sido desarrollada por **Félix Jiménez Gonzalez** y **Vincenzo Parretta**

Para el desarrollo de este trabajo, se seguirá en un principio las pautas de un proyecto típico guiado por KDD (Knowledge Discovery in Database), seleccionando el conjunto de datos que se detallará más adelante, realizando las operaciones de procesamiento y transformaciones para poder adaptar el conjunto de datos a las operaciones a realizar en el entorno, se aplicará el data mining a través de distintos procedimientos, dependiendo del tipo de aprendizaje y técnica que se esté abarcando, y por último se realizará una interpretación de los resultados obtenidos.

Cabe destacar que el entorno de trabajo utilizado para la realización de ambos análisis se trata de **RStudio** junto con librerías determinadas dependiendo de la necesidad de las mismas en ciertos puntos del trabajo (se mencionarán si es necesario). Por último, toda la información relativa al proyecto, incluido el código de la misma, se puede encontrar en el siguiente repositorio de GitHub, en el cual se ha llevado a cabo el desarrollo del proyecto: https://github.com/josaloroc/YourPosition

A continuación, en las siguientes secciones del proyecto se detallará el desarrollo de cada uno de los tipos de análisis propuestos, redactados por cada una de las parejas presentadas en los párrafos anteriores. En cada una de las correspondientes secciones se explicará como se ha orientado y guiado el análisis, herramientas utilizadas, detalles sobre los planteamientos, justificaciones sobre parámetros e informe o interpretación de los resultados obtenidos.

Sin embargo, previamente, se realizará una breve introducción al conjunto de datos escogido para el problema, haciendo adicionalmente una breve introducción y puesta al contexto sobre el ámbito de la temática a tratar.

2. Conjunto de datos seleccionado

El conjunto de datos seleccionado sobre el que trabajar reúne la información y estadísticas del desempeño de todos los jugadores de fútbol de ligas europeas más importantes durante la temporada 2021-2022. Este conjunto de datos abarca aproximadamente 3000 filas, donde cada una de ellas representa a un jugador, y 143 columnas, que sirven para definir cada una de las estadísticas de los jugadores para medir su desempeño.

El equipo decidió seleccionar este conjunto debido a la amplia cantidad de información que abarca, que puede llegar a ser muy beneficiosa para este proyecto, además de la exactitud y la definición de las distintas estadísticas que comprende el conjunto de datos, cuyo tratamiento para el proyecto, junto con los conocimientos básicos de fútbol, podría llegar a resultar en algo bastante interesante.

Por otro lado, lo más importante de este conjunto de datos es la representación en otra columna del conjunto de la posición del jugador, por lo que esto sirve como etiquetas para la tarea de clasificación como aprendizaje supervisado. Otro detalle interesante de estas etiquetas, consiste en como vienen distribuidas las distintas posiciones de los jugadores. Más allá de las categorías principales: portero, defensa, mediocentro y delantero, hay muchas más posiciones determinadas dentro de estas categorías según la función que desempeñan. Si el conjunto de datos abarcara las etiquetas con posiciones exactas, sería mucho más complejo, pero la distribución de las posiciones viene etiquetada de una forma que puede resultar beneficiosa a la hora del análisis de resultados. A continuación exponemos las etiquetas que comprende el problema:

- Para los delanteros: **FW** (del inglés, *forward*)
- Para los mediocentros: **MF** (del inglés, *mid-field*)
- Para los defensas: **DF** (del inglés, *defender*)
- Para los porteros: **GK** (del inglés, *goalkeeper*)
- Mezcla de posiciones principales para aquellos jugadores que puedan jugar en más de una posición. Esto viene determinado por la mezcla de los anteriores términos, por ejemplo, **MFFW** puede ser un mediocentro que puede jugar de delantero, o al contrario, **FWMF**, un delantero que puede jugar como mediocentro. Existen hasta 10 combinaciones posibles en todo el conjunto de datos.

La distribución de las posiciones en etiquetas de la anterior forma puede ayudar a enriquecer los resultados del problema, de forma que, por ejemplo, si un defensa ha sido clasificado como defensa-mediocentro, aunque la clasificación no haya sido totalmente acertada, sí que puede con cierta lógica encajar el jugador en dicha posición, ya que realmente hay similitudes entre las funciones que desempeñan ambas posiciones y perfectamente el jugador podría encajar dentro de esa posición. Consideramos que los mayores errores se pueden dar cuando, por ejemplo, un defensa es clasificado como delantero puro, cosa que es mucho más improbable que ocurra en el propio deporte.

Respecto a las columnas del conjunto de datos, se registran estadísticas muy complejas, más allá de las recurrentes que se suelen mostrar en determinadas fuentes al redactar resúmenes de partidos o jugadores. Debido a su gran extensión y por la facilidad de entendimiento de las columnas, se puede ver toda la información relativa a dichas estadísticas en la página correspondiente al conjunto de datos: https://www.kaggle.com/datasets/vivovinco/20212022-football-player-stats

De aquí surge otra de las posibles tareas a realizar en la extracción de conocimiento de este trabajo a través de los resultados obtenidos: la importancia de las variables seleccionadas. El conjunto de datos comprende de una gran cantidad de información, no obstante, es probable que no todas las estadísticas sean igual de relevantes o aporten una información valiosa para el entrenamiento. Se tratará de obtener las variables más importantes relacionándolo con el propio deporte a través de explicaciones basadas en dicho conocimiento.

3. Aprendizaje supervisado: clasificación

En esta sección se presentará el trabajo realizado por **José Luis Alonso Rocha** y **Fernando Alonso Martín**. Se abarcará todo el procedimiento, lógica, diseño y justificación, tanto de la propuesta como las decisiones tomadas a lo largo del desarrollo de las distintas fases del trabajo de análisis de datos. Para una mayor claridad, se tratará de acompañar con fotografías o tablas con base a la información obtenida de RStudio, de forma que, elementos como los resultados, sean más visibles y fácil su comparativa.

3.1. Introducción al problema: inicialización de datos y paquetes

A continuación, procedemos a introducir las herramientas y paquetes que se han utilizado en este estudio y análisis, de forma que nos sirven como mayor o menor apoyo en determinados puntos del proyecto. Agrupamos y presentamos los paquetes de la siguiente forma:

- **Caret** y **Tidyverse**. El primero de ellos, se ha usado para el procesamiento y entrenamiento de los modelos, suponiendo el paquete más importante en el proyecto, y, por otro lado, tidyverse, se ha empleado en determinados puntos para realizar determinadas operaciones con los datos que se manejaban en el proyecto.
- **Parallel** y **doParallel** para la división en paralelo de los trabajos a procesar por RStudio al ejecutar determinadas funciones. Esto sirve para reducir el tiempo en el entrenamiento de los modelos debido al alto coste computacional que supone para nuestros equipos realizar determinadas operaciones con el extenso conjunto de datos elegido.
- **MLmetrics**, **randomForest** y **ggplot2** paquetes auxiliares para el entrenamiento de modelos que siguen determinados métodos. Por otro lado, ggplot2 nos permitirá visualizar de forma determinada diversos resultados, por lo que nos permitirá facilitar comparaciones.

En siguiente lugar, dentro de esta sección se inicializa también a través de las librerías doParallel y Parallel, los distintos clústers en los que dividir los trabajos de ejecución de las distintas partes de nuestro proyecto, para que se ejecuten con un mayor rendimiento. Esto se inicializa al principio para que todas las funciones pueden aprovecharse de sus beneficios.

Finalmente, se realiza la lectura del conjunto de datos sobre el que se realizarán operaciones previas sobre sus datos para poder operar adecuadamente.

3.2. Preprocesado

A continuación, en el siguiente apartado se detallarán los arreglos que se han aportado al conjunto de datos previamente al uso de los mismos para el proceso de data mining en general. Uno de los aspectos más importantes con los que se ha tenido que lidiar en el trabajo es el coste computacional de las operaciones debido a la extensión del conjunto de datos. Por ello, se tomaron una serie de decisiones buscando optimizar la eficiencia de la ejecución, sin que llegue a provocar posibles efectos perjudiciales para el entrenamiento e intentando encajarlo con arreglos al propio conjunto que puedan llegar a beneficiarnos. En la siguiente lista se especifica las operaciones aplicadas:

- **Recorte del conjunto de dataset para jugadores que hayan jugado al menos 10 partidos**. Esto se debe a que los propios datos registran jugadores con muy pocos partidos jugados y por ende, con unas estadísticas que no pueden llegar a ser realmente representativas o que no aportan suficiente valor a la información de su posición. Debido a que añadían ruido al propio conjunto de datos con valores poco relevantes y aumentaba mucho el coste de ejecución, se llegaron a recortar aproximadamente 1000 jugadores, dejando un total de 2000 filas, mejorando un poco los resultados y notablemente el coste de ejecución.
- **Eliminación de diversas columnas**. Datos como nombre del jugador, club, fecha de nacimiento, edad, etc. son totalmente irrelevantes para nuestra variable objetivo. Hay al menos 10 columnas que se han eliminado totalmente.

- **Transformación de los tipos de variables a tratar**. La lectura del conjunto datos provocaba que, por defecto, muchos de los tipos de los datos de columnas se leyeran en formatos inadecuados, por lo que se tuvieron que realizar determinadas operaciones para poner cambiar el tipo de datos, además del tratamiento de valores perdidos. La columna objetivo, posición, se ha tratado como "factor", el resto, como numéricas.

Finalmente, una vez modificado el conjunto de datos inicial, se procede a utilizar operaciones básicas para crear la partición de datos que usaremos para el entrenamiento y generación del modelo: un conjunto de entrenamiento y otro conjunto de prueba. Se obtienen las columnas de la variable objetivo para cada uno de los casos, y se comprueba antes con una función determinada el número de valores perdidos en cada columna. Tras haber tratado con ellos con lo mencionado en el párrafo anterior y obtener un número de 0 valores perdidos en todas las columnas, se procede con el entrenamiento.

3.3. Entrenamiento

En la siguiente sección se tratará el contenido más extenso que supone el núcleo principal del trabajo, compuesto por los distintos entrenamientos configurados y llevados a cabo a través del uso de la herramienta **Caret** para poder analizar, comparar y estudiar los distintos modelos resultantes para la selección de aquel más óptimo para el problema que se trata de abarcar.

En primer lugar, para una mejor comprensión de lo redactado en posteriores sub apartados, resulta necesario introducir el planteamiento de la estructura del análisis realizado. El objetivo principal impuesto por nuestro subgrupo se trataba de la clasificación o etiquetado a través de la predicción respecto a posiciones de jugadores de fútbol a partir de sus estadísticas referentes al rendimiento de una temporada. Se trata de un problema de clasificación multiclase, ya que disponemos de hasta aproximadamente 10 etiquetas distintas respectivas a las posiciones de los jugadores con las que se les puede etiquetar en la predicción. Por ello, se decidió realizar una búsqueda de información previa para reunir conocimiento respecto a métodos y algoritmos a emplear de los que disponía **Caret** recomendados para abordar este tipo de problemas.

Una vez reunidos los principales métodos/algoritmos y sus derivados con los que poder empezar a entrenar y generar modelos, se planteó la organización de la fase de entrenamiento de forma que se estableciera un punto a través del cual desarrollar el entrenamiento con el objetivo de conseguir un estudio y análisis para obtener un modelo óptimo. Este planteamiento consiste simplemente en el entrenamiento y generación de multitud de modelos, a través de diversas parametrizaciones y configuraciones si lo requiere, ya sea manual (si aplica) o automático, y probando cada una de las posibles configuraciones con distintos tipos de controladores para el entrenamiento.

La idea fundamental se basa en obtener una amplia gama de modelos con los que ir comparando y seleccionando entre ellos de forma que se maximize nuestro valor objetivo: **Accuracy**. Para conseguir esto, en algunos casos se ha hecho uso de pequeñas funcionalidades obtenidas a través de búsquedas en internet, para conseguir una mayor precisión y probar nuevas alternativas en los entrenamientos establecidos. El resultado de la idea de esta fase, sería obtener el mejor modelo por cada tipo de algoritmo/método empleado en los entrenamientos, para posteriormente, en la fase de resultados, comparar la eficiencia entre los mejores clasificadores de cada algoritmo empleado. A continuación, mostramos un esquema de la fase de entrenamiento general, para una mayor claridad y comprensión de lo descrito en este párrafo:

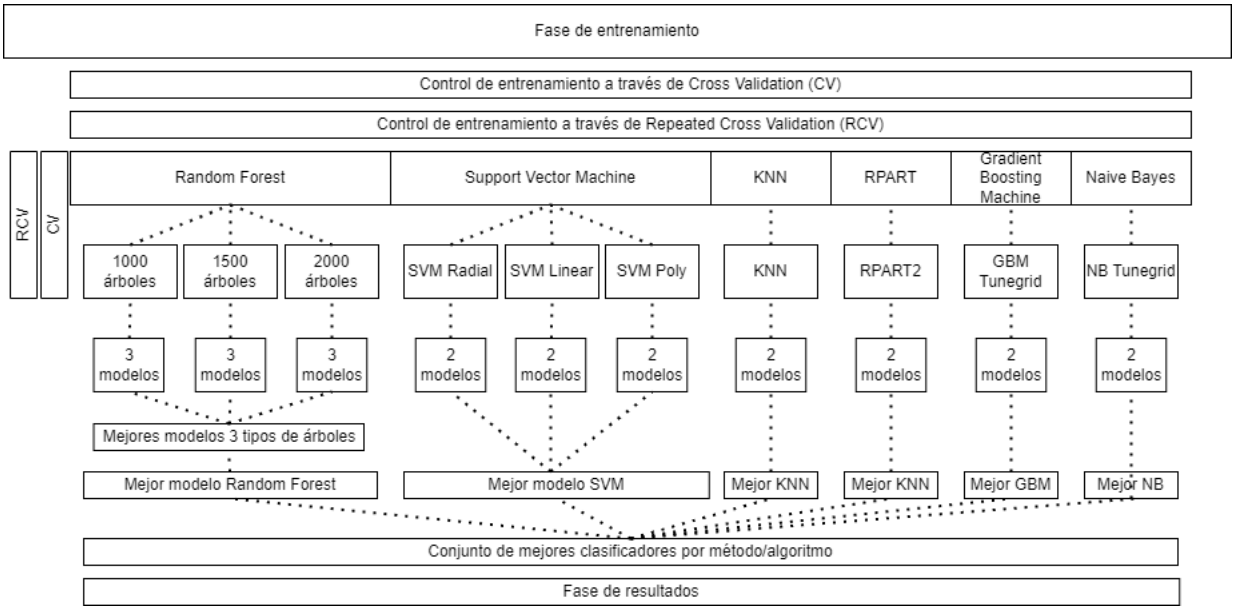


Figura 1: Estructura a alto nivel del planteamiento del entrenamiento

3.3.1. Inicio del entrenamiento

Como paso previo a la generación de modelos, declaramos los parámetros de las funciones *trainControl* que se emplearan para controlar los entrenamientos de los distintos modelos de forma alternativa, de forma que se puedan conseguir más pruebas de distinto calibre. Se han ajustado en total 4 funciones de control de entrenamiento distintas, teniendo siempre en cuenta el coste computacional que supone ajustar estas opciones, ya que, debido a varias pruebas que se hicieron con determinados parámetros, provocaba un alto consumo de memoria en RStudio y nunca llegaba a finalizar el entrenamiento en los equipos del subgrupo. Por lo tanto, se han ajustado las siguientes funciones:

- **Cross-validation**. Ajustado tanto para todos los algoritmos a emplear en los entrenamientos, como para el algoritmo **GBM** por separado. Este último hecho se debe al alto coste computacional que supone emplear dicho algoritmo con nuestro conjunto de datos. En común, se han establecido 10 iteraciones y se ha activado el procesamiento paralelo para aumentar la eficiencia de la ejecución. Adicionalmente, se ha establecido que el ajuste de parámetros sea aleatorio, excepto en el caso de **BGM** que se ha dejado por defecto.
- **Repeated Cross-Validation**. Establecido siguiendo la misma pauta que el caso anterior. Simplemente, se ha añadido exclusivamente como número de repeticiones, 10, para todos los algoritmos, excepto para **BGM**, cuyo número se ha establecido a 3, por motivos de coste computacional.

3.3.2. Modelos entrenados con Random Forest

En primer lugar, se han generado los modelos a través del algoritmo de *Random Forest*. Se ha dividido la generación de modelos en 3 partes, cada una de ellas, conteniendo hasta 3 entrenamientos, eligiendo al final en cada parte el modelo con mayor *Accuracy*, resultando en 3 modelos finales, entre los cuales se comparan y se obtiene el mejor siguiendo la misma métrica. Cada una de dichas partes está diferenciada por el número de árboles con el que se genera el entrenamiento del modelo, respectivamente: 1000, 1500 y 2000 árboles.

Antes de empezar los entrenamientos, calculamos uno de los parámetros que será útil en cada uno de ellos: **mtry**. Tanto el número de árboles como el parámetro **mtry** son dos de las características fundamentales ajustar para el algoritmo de *Random Forest*. Esta última variable sirve para establecer que el número de variables se recoja aleatoriamente para ser muestreado en cada momento de la división del árbol. Gracias a unas determinadas funciones que procesan el conjunto datos y determinan el valor de **mtry** adecuado, Caret puede realizar los entrenamientos de una forma mucho menos costosa debido

a que solo se centra en el valor que se ha introducido. Dicho valor se calcula para los entrenamientos de 1000, 1500 y 2000 árboles.

A continuación, exponemos los tipos de modelos entrenados y los resultados. Para cada parte se han generado 3 entrenamientos:

- Entrenamiento básico, con el número de árboles determinado por la parte y el valor de *tuneGrid* ajustado al calculado anteriormente.
- Otros dos entremientos más avanzados, añadiendo el control del entrenamiento, tanto con **Cross-Validation** y **Repeated Cross-Validation**

Modelo	**Accuracy**
RF 1000 Trees	0.785
RF 1000 Trees RCV	0.790
RF 1000 Trees CV	0.793

Cuadro 1: Comparación de la precisión respecto a los tres modelos entrenados con Random Forest para 1000 árboles

En la tabla superior se ha escogido para cada uno de los entrenamientos que realiza **Caret** en la generación de modelos, aquel que mayor accuracy presenta debido a los ajustes automáticos que realiza el mencionado paquete en el entrenamiento del modelo para buscar el mejor. Esto da como resultado 3 modelos finales, que, como se puede comprobar en la tabla anterior, el modelo controlado con *Cross-Validation* es el que presenta mejores resultados.

Continuamos con los resultados para la generación de modelos con las mismas condiciones que el apartado anterior, pero esta vez para 1500 árboles, ajustando como es debido el valor de **mtry** calculado previamente en pasos anteriores.

Modelo	**Accuracy**
RF 1500 Trees	0.788
RF 1500 Trees RCV	0.790
RF 1500 Trees CV	0.792

Cuadro 2: Comparación de la precisión respecto a los tres modelos entrenados con Random Forest para 1500 árboles

Nuevamente, para este caso, el mejor modelo se ha obtenido de la mano del entrenamiento controlado a partir de *Cross-Validation*. Los resultados respecto a la precisión difieren muy poco entre sí, en comparación con los resultados de la primera tabla.

Finalmente, se muestran los resultados para la generación de modelos para 2000 árboles y su valor correspondiente del **mtry**.

Modelo	**Accuracy**
RF 2000 Trees CV	0.780
RF 2000 Trees	0.784
RF 2000 Trees RCV	0.788

Cuadro 3: Comparación de la precisión respecto a los tres modelos entrenados con Random Forest para 2000 árboles

Se pueden observar unos resultados ligeramente similares al resto de modelos generados en apartados anteriores, con un mínimo descenso del valor de *accuracy*. En este caso, el mejor modelo resultante ha sido aquel cuyo control del entrenamiento ha sido llevado a cabo por *Repetead Cross-Validation*

Finalmente, comparamos los mejores modelos resultantes en la tabla final, y seleccionamos para el procedimiento final, el de mayor *Accuracy*.

Modelo	**Accuracy**
RF 2000 Trees RCV	0.788
RF 1500 Trees CV	0.792
RF 1000 Trees CV	0.793

Cuadro 4: Comparación de la precisión respecto a los mejores modelos de cada parte

3.3.3. Modelos entrenados con Naive Bayes

En segundo lugar, el entrenamiento de los modelos se ha realizado a través de *Naive Bayes*. El coste computacional ejecutando estos entrenamientos ha resultado ser mucho más bajo en comparación con el caso anterior, con *Random Forest*, cuyos tiempos de ejecución eran bastante altos, incluso usando el procesamiento paralelo.

Se ha seguido una metodología similar al caso anterior, pero mucho más sencilla, debido a que no hay tanta variedad de parámetros relevantes en *Naive Bayes* con los que generar modelos diferenciadores a diferencia del apartado anterior con el número de árboles. Por ello, se han establecido simplemente dos pruebas, cada una de ellas controladas con **CV** o **RCV**, estableciendo un control manual de los parámetros para la mejor selección del modelo. Los resultados han sido los siguientes:

Modelo	**Accuracy**
Naive Bayes RCV	0.720
Naive Bayes CV	0.722

Cuadro 5: Comparación de la precisión respecto a los modelos generador por Naive Bayes

Como se puede comprobar en la tabla de resultados, se ha obtenido un mejor resultado para el modelo generado con *Cross-Validation* respecto al *Repeated Cross-Validation* por una diferencia mínima. En comparación con los resultados vistos en los modelos entrenados a partir de *Random Forest*, sí que se puede contemplar una diferencia más notoria de casi hasta un 8 %.

3.3.4. Modelos entrenados con Support Vector Machine

En tercer lugar, continuamos el entrenamiento y generación de modelos a través del algoritmo *Support Vector Machine*. Se han desarrollado numerosos entrenamientos debido a las distintas variaciones existentes de dicho algoritmo, por lo que, junto a los distintos tipos de controles de entrenamientos establecidos, se ha aprovechado para poder obtener más posibles modelos con los que establecer comparaciones.

Además de las anteriores opciones, adicionalmente se ha añadido una función de preprocesamiento en el método de entrenamiento de Caret: **Centrado** para extraer la media de las variables predictoras y **Escalado** para dividirla por la desviación estándar.

También, se ha alternado entre un ajuste de parámetros manual con *tuneGrid* o estableciendo un número bajo para *tuneLength* para que se controlen las combinaciones de parámetros con un número bajo, de forma que no afecte demasiado al coste computacional, ya que a diferencia de *Naive Bayes*, este algoritmo consume más recursos, pero sin llegar a los límites de *Random Forest*. A continuación, exponemos las variaciones del algoritmo empleadas para el entrenamiento y acto seguido los resultados del mismo: SVM Lineal, SVM No-Lineal (Radial) y SVM Polinomial.

Modelo	Accuracy
SVM Poly RCV	0.746
SVM Linear CV	0.755
SVM Linear RCV	0.760
SVM Linear CV Ajustado manualmente	0.764
SVM Linear RCV Ajustado manualmente	0.767
SVM Radial CV	0.776
SVM Radial RCV	0.781
SVM Poly CV	0.788

Cuadro 6: Comparación de la precisión respecto a los modelos generados por Support Vector Mechine

Los resultados anteriores muestran una determinada cantidad de resultados con un rango de precisión adecuado. Los valores más bajos superan a los de *Naive Bayes*, mientras que los más altos pueden llegar a igual a los de *Random Forest*. Nuevamente, se consiguen unos valores decentes respecto al entrenamiento de los modelos, por lo que se puede esperar resultados adecuados en las pruebas con predicciones con el conjunto de test.

3.3.5. Modelos entrenados con Gradient Boosted Machine

En cuarto lugar, presentamos los modelos generados a través del algoritmo *Gradient Boosted Machine*. La aproximación que se ha tomado con el entrenamiento de estos modelos ha sido parecida a la de *Naive Bayes*. En un principio, se había contemplado adicionalmente el entrenamiento de modelos a través del ajuste manual de parámetros, sin embargo, fue en este tipo de pruebas cuando se llegó a concluir el alto coste computacional que suponía el empleo de este algoritmo junto con nuestro conjunto de datos. Sin lugar a dudas, **GBM** supone un gasto de recursos y memoria del equipo mucho mayor que la que presenta *Random Forest*, llegando a tardar alrededor de 5 minutos en los entrenamientos con configuraciones normales. Para otras pruebas que no se han llegado a contemplar, el entrenamiento no se pudo llegar a concluir, debido a que nunca se completaba el mismo y se debía de detener.

Modelo	Accuracy
GBM CV	0.776
GBM RCV	0.780

Cuadro 7: Comparación de la precisión respecto a los modelos generados por Gradient Boosted Machine

Los resultados obtenidos siguen la misma línea respecto a los resultados anteriores. De forma general, nuevamente se tratan de resultados muy buenos, aproximadamente un 80 % de accuracy, del que probablemente se puedan obtener resultados favorables en la última fase de pruebas con el conjunto de test. En comparación con el resto de algoritmos hasta el momento, se encuentra entre la media de los resultados, rozando casi el resultado más alto de *Random Forest*.

3.3.6. Modelos entrenados con K-nearest neighbors

En quinto lugar, presentamos los modelos entrenados con **KNN**. Nuevamente, al contrario que en otros casos, no se han encontrado demasiados elementos o parámetros conocidos con los que poder ajustar el entrenamiento, por lo que se ha optado por realizar entrenamientos sencillos como el caso de *Naive Bayes* a partir de diversos parámetros que ya conocíamos anteriormente. Se ha establecido un *tuneLength* medianamente alto, ya que el coste computacional lo permitía, y a su vez, nuevamente, para facilitar los cálculos con este algoritmo, se ha vuelto a hacer uso de la función de *preProcess* para normalizar los valores en un rango determinado.

Modelo	Accuracy
KNN CV	0.764
KNN RCV	0.774

Cuadro 8: Comparación de la precisión respecto a los modelos generados por K-nearest neighbors

Se han obtenido unos resultados decentes, en un punto medio entre todos los algoritmos vistos hasta el momento. Más adelante comprobaremos la efectividad en las predicciones. No llega a tener resultados como los de *Naive Bayes*, pero sin alcanzar al resto de algoritmos que tienen resultados más notables.

3.3.7. Modelos entrenados con CART

Finalmente, se introducen los resultados obtenidos para el entrenamiento a través de CART haciendo uso del método *rpart2* con Caret. Siguiendo la misma línea que para los últimos entrenamientos, no se han encontrado parámetros exclusivos y relevantes con los que personalizar manualmente el entrenamiento del modelo con CART, por lo que se ha optado simplemente por hacer dos entrenamientos con ajustes básicos, teniendo en cuenta siempre el coste computacional al tratarse de operaciones con árboles y el amplio tamaño del conjunto de datos.

Modelo	**Accuracy**
RPART RCV	0.756
RPART CV	0.758

Cuadro 9: Comparación de la precisión respecto a los modelos generador por CART

Los resultados finales obtenidos nuevamente se encuentran en un punto intermedio respecto a los que han mostrado los anteriores algoritmos. Hay una diferencia mínima para lo obtenido entre *RCV* y *CV*. Estos resultados no son notables como para los casos presentados con *Random Forest*, pero casi iguala o supera determinados casos como *Naive Bayes* o *KNN*.

3.3.8. Resultados finales de la fase de entrenamiento

En el siguiente apartado, tras haber puesto en práctica los entrenamientos planteados en los primeros apartados, procedemos a la comparación final de todos los resultados obtenidos de los anteriores casos. Como se mencionaba en los primeros apartados de planteamiento y estructura del problema y se ve reflejado en la primera figura donde se resume a nivel general todo el procedimiento, tras la comparación y selección de modelos individualmente por algoritmo empleado para el entrenamiento, se procede a realizar la comparativa general de los resultados respecto a los mismos modelos de forma que se pueda concluir la fase de entrenamiento y continuar con dichos resultados a la fase de pruebas para extraer las conclusiones y vincularlo con el conocimiento actual del problema.

Los resultados de las fases anteriores, seleccionando el mejor modelo por cada caso y ordenados desde el peor hasta el mejor modelo, son los siguientes:

Modelo	**Accuracy**
NB CV	0.722
RPART CV	0.758
KNN RCV	0.774
GBM RCV	0.780
SVM Poly CV	0.788
RF 1000 Trees CV	0.793

Cuadro 10: Comparativa final entre todos los mejores modelos obtenidos en la fase de entrenamiento

Lo mostrado en la tabla anterior son los máximos valores de *accuracy* que se han podido lograr obtener para cada tipo de entrenamiento empleando un método distinto. Generalmente, se han conseguido desde unos resultados aceptables hasta otros bastante favorables, como llega a ser en el mejor de los casos. Los resultados, como se puede comprobar, se encuentran entre un rango de 70 %-80 % de *accuracy* lo que supone unos resultados favorables siguiendo el planteamiento propuesto al inicio del documento.

La obtención de estos resultados se complementa con casos no registrados en este documento que se contemplaron en las primeras fases iniciales de entrenamiento, donde se realizan pruebas iniciales

con los algoritmos, sin pautas específicas para los ajustes y parametrización, de forma que se pudiera ir estableciendo un punto de partida. En estas pruebas iniciales realizadas, había casos en los que los resultados no eran tan favorables como los mostrados anteriormente. Este caso era más notable en los modelos entrenados con **SVM** y **Naive Bayes**, cuyos resultados iniciales respecto al *accuracy* rondaban entre 55 %-70 %.

El ajuste y parametrización realizado ha ayudado a conseguir resultados más óptimos en determinados casos, profundizando mucho más en los entrenamientos y planteándolos de distinta forma. Cabe destacar, que en uno de los entrenamientos realizados con **RF** se llegó a obtener hasta 81 % de *accuracy*, siendo este el mejor resultado obtenido en todo el proyecto de aprendizaje supervisado, sin embargo, no se ha registrado en los resultados de este documento porque la última iteración sobre esos entrenamientos ha resultado ser la mostrada en su respectivo apartado.

Para asegurar la integridad y coherencia de los resultados, como se deduce respecto a lo comentado en el párrafo anterior, no nos hemos limitado a hacer una simple ejecución de los modelos de entrenamiento: se han realizado varias ejecuciones de todo lo planteado, obteniendo en todos los casos resultados similares con mínimas variaciones a los mostrados en este informe de resultados. No obstante, si es cierto que esas mínimas diferencias han podido provocar la obtención de mejores modelos respecto a los mostrados en los casos anteriores (por ejemplo, para **SVM**, el mejor resultado era para **SVM Radial CV** en algunos casos), pudiendo a llegar a cambiar ligeramente la clasificación final mostrada. Sin embargo, todas las ejecuciones realizadas muestran resultados muy similares a los mostrados en este informe, así como una clara tendencia a determinar como mejor modelo a *Random Forest*.

Como reflexión respecto a los resultados, desconocemos si se pueden alcanzar resultados más favorables de los obtenidos hasta el momento. En nuestro caso, no se ha podido obtener de forma consistente unos resultados superiores a 81 % de *accuracy*. Esto se debe a una de nuestras mayores limitaciones respecto a los entrenamientos: el coste computacional de las operaciones debido al procesamiento de **Caret** con la función *train* junto con la gran extensión de nuestro conjunto de datos. Este hecho ha provocado el recorte de diversos entrenamientos planteados para muchos de los métodos o algoritmos planteados. Se intentaron realizar dichas pruebas finalmente suprimidas, pero esto llegaba a suponer tal consumo de memoria para RStudio que, ni aun utilizando el procesamiento paralelo de operaciones, lograba poder completar las operaciones.

Probablemente, se puedan obtener resultados ligeramente más positivos de los mostrados anteriormente, aunque, con todas las operaciones que nuestro análisis ha cubierto, con todas las configuraciones y ajustes que nos podíamos permitir, el único planteamiento que se nos ocurre para poder intentar mejorar los resultados es realizar entrenamientos más costosos.

3.4. Análisis de las variables post-entrenamiento

En el siguiente apartado, realizaremos un breve análisis haciendo uso de una de las funciones que nos ofrece el paquete Caret junto con los resultados de los apartados anteriores. Se trata del cálculo, estudio y obtención de las variables más importantes o significativas para los entrenamientos procesados en el apartado anterior a la hora de determinar las distintas posiciones de un jugador. Estos resultados pueden llegar a ser realmente interesantes, ya que aquí se puede descubrir los aspectos más importantes que definen la posición de un jugador en el fútbol, así como tendencias y patrones para ver que más factores son los que tienen en común dichas posiciones.

Se ha calculado la importancia de las variables para todos los casos de entrenamiento registrados en la tabla final. Se obtienen dos tablas de resultados distintos entre todos los algoritmos y los resultados respecto a la importancia de variables suele ser muy similar, por lo que apenas hay diferencias. Además, observando la precisión en los entrenamientos y los posibles buenos resultados futuros en las predicciones, podemos tener cierto nivel de seguridad sobre la confianza de estos resultados que se muestran a continuación. Establecemos dos tipos de tabla de muestra de resultados, extraídas de los mejores modelos obtenidos en el apartado anterior (tabla de importancia general, de *Random Forest*, y la tabla de importancia de variable para posiciones, extraída de *SVM Poly*):

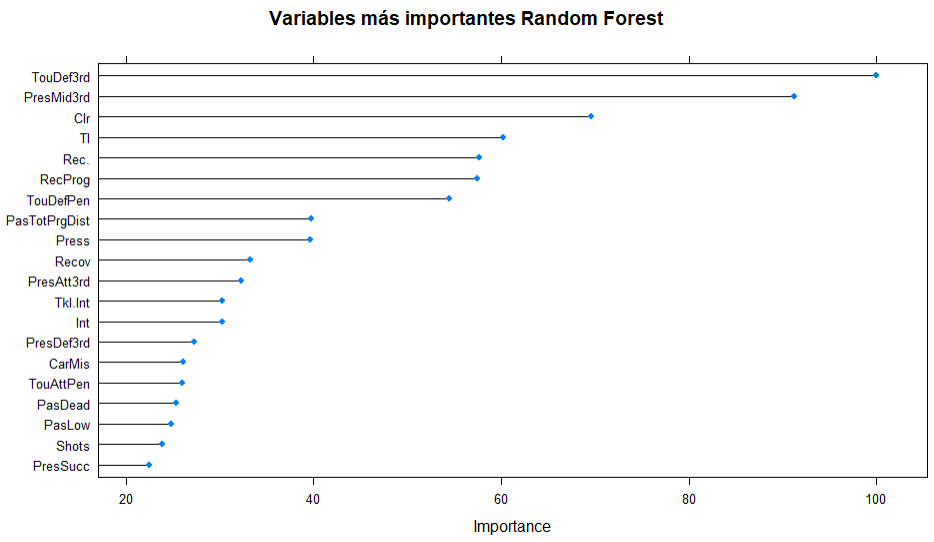


Figura 2: Importancia de variables general. Extraída del mejor modelo de Random Forest.

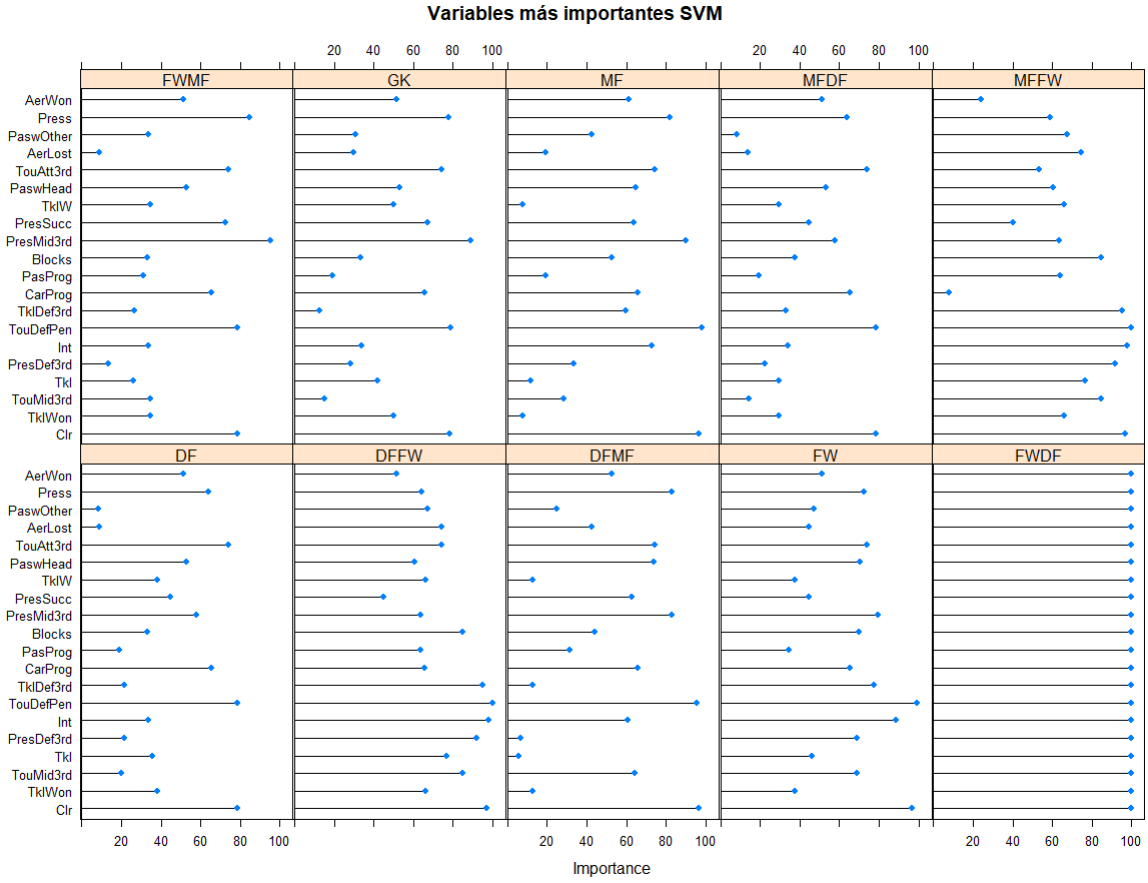


Figura 3: Importancia de variables por posición. Extraída del mejor modelo de SVM.

A continuación, exponemos una relación general de los conceptos presentados en las imágenes anteriores junto con el conocimiento base del problema a tratar. En un principio, lo más notable que nos llamó la atención al obtener los resultados respecto a las variables más importantes se trataba de los las estadísticas que se habían interpretado como más relevantes y que mayor ganancia de infor-

mación aportaban al entrenamiento, pues pensábamos en un principio que dicho ranking de variables más importantes estaría compuesto por las estadísticas más simples y generales del fútbol, aquellas características más definitorias de cada una de las posiciones, como por ejemplo, el número de goles marcados, faltas y fueras de juego cometidas, etc.

Sin embargo, los resultados, han sido totalmente distintos a lo que esperábamos, hecho que no tiene por qué ser negativo, sino que aumenta la curiosidad y fomenta la ganancia de conocimiento para las conclusiones extraídas. Hemos realizado un breve análisis sobre el ranking obtenido, deduciendo la relación de las variables obtenidas en dicha clasificación, justificándolo a través del sentido del propio deporte. Debido a que se tratan un total de 20 variables, comentaremos algunas de las más notables:

Recomendado, para una mejor orientación respecto al significado de las variables, consultar la siguiente web: https://www.kaggle.com/datasets/vivovinco/20212022-football-player-stats

- Variables relacionadas con la localización de los jugadores: **TouDef3rd, PresMid3rd, PresAtt3rd, PresDef3rd**. Estas variables, dependiendo del nombre, suelen determinar el número de toques realizados al balón o el tiempo de presión ejercida a la hora de defender para recuperar la pelota en un determinado tercio del campo.

  Dichos valores variarán bastante dependiendo de la posición del jugador, además, como se puede ver en el propio nombre de diversas variables, tiene sentido que sean las que más información aporten a la posición del jugador debido a que precisamente registran una estadística (presión) para cada una de las 3 posiciones fundamentales.

- Variables relacionadas con acciones relativas de posiciones de los jugadores: **Clr, Rec, Press, Recov, Shots, Int**. Dichas variables suelen estar relacionadas con la tendencia a que determinadas posiciones de jugadores realicen con mucha más frecuencia que otros, debido al rol que desempeñan jugando en su posición correspondiente.

  Por ejemplo, **Clr** (despejes) , **Press**, (presión) y **Recov** (recuperación de balón) son acciones estrechamente vinculadas con aspectos defensivos del juego, por lo que existirá una clara diferencia entre los valores que registren los defensas respecto a los que tengan otros jugadores. En menor medida, la acción de Shots (disparos) puede ser determinante, ya que es una acción vinculada a los delanteros, por la naturaleza ofensiva de esta acción.

  Sin embargo, esto último no es totalmente cierto, ya que aunque es más probable que un delantero registro un mayor número de tiros a diferencia de otras posiciones, no existe un real diferencia ya que cualquier jugador puede realizar un disparo por determinadas acciones en el juego (debido al lanzamiento de una falta, un remate en un saque de esquina donde se concentran todos los jugadores en un equipo, etc. Esto fomenta que la distribución de tiros no sea tan diferenciadora).

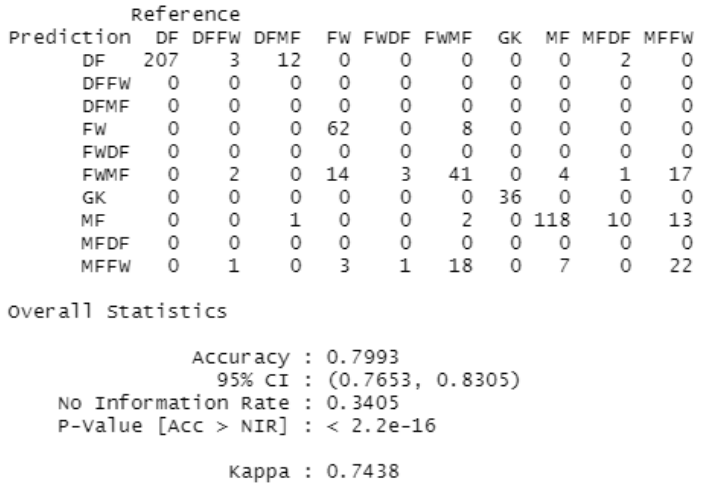
- Otras variables del juego: como **PressSuc** (tiempo de presión ejercida antes de recuperar el balón) mayormente relacionado con aspectos defensivos del juego, o todas aquellas variables relacionas con pases específicos (**PasDead** o **PasLow**) que son más probables a ser realizados por posiciones específicas dependiendo de la naturaleza del tipo de pase registrado.

3.5. Resultados y relación con el conocimiento del problema

Finalmente, tras las fases anteriores con el entrenamiento y selección de los modelos, procedemos a poner a prueba su eficacia y precisión estimada a través de las predicciones empleando los mismos modelos y la segunda división del conjunto de datos: el relativo a la fase de pruebas del problema. En este apartado, se calculará la matriz de confusión a través de lo especificado previamente, de forma que nos pueda ayudar a mostrar los resultados del entrenamiento y la precisión del clasificador de una forma más clara mientras se relacionan los mismos resultados con el ámbito del problema.

El planteamiento de todas las matrices de confusión son iguales, por ello, se mostrarán aquellas con los mejores resultados obtenidos a través de los modelos anteriores, junto con sus estadísticas

más relevantes. Por último, como todos los datos (etiquetas/posiciones) que aparecen en la matriz son iguales o tienen la misma naturaleza, cualquiera de las mostradas se puede utilizar como referencia para poder analizar y explicar los resultados relacionándolo con el propio deporte, por lo tanto, en este caso, utilizamos el esquema general mostrado por las matrices de confusión expuestas. A continuación, mostramos los resultados de las matrices relacionadas de los 3 modelos con mayor *accuracy* obtenidos en los apartados anteriores:



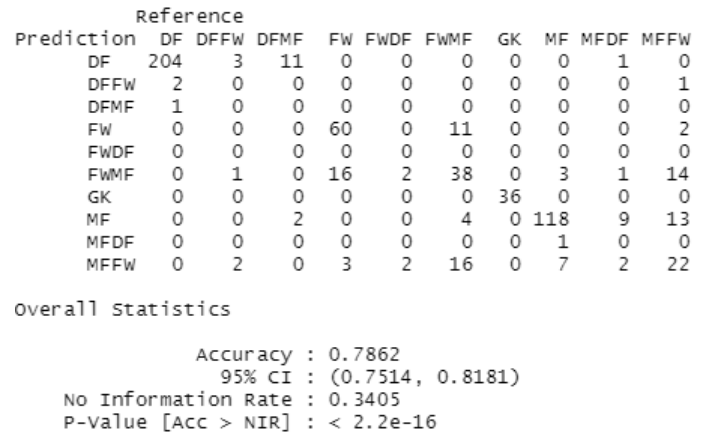
```
          Reference
Prediction  DF DFFW DFMF  FW FWDF FWMF  GK  MF MFDF MFFW
      DF   207    3   12   0    0    0   0   0    2    0
      DFFW   0    0    0   0    0    0   0   0    0    0
      DFMF   0    0    0   0    0    0   0   0    0    0
      FW     0    0    0  62    0    8   0   0    0    0
      FWDF   0    0    0   0    0    0   0   0    0    0
      FWMF   0    2    0  14    3   41   0   4    1   17
      GK     0    0    0   0    0    0  36   0    0    0
      MF     0    0    1   0    0    2   0 118   10   13
      MFDF   0    0    0   0    0    0   0   0    0    0
      MFFW   0    1    0   3    1   18   0   7    0   22

Overall Statistics

               Accuracy : 0.7993
                 95% CI : (0.7653, 0.8305)
    No Information Rate : 0.3405
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7438
```

Figura 4: Matriz de confusión del clasificador entrenado con Random Forest.



```
          Reference
Prediction  DF DFFW DFMF  FW FWDF FWMF  GK  MF MFDF MFFW
      DF   204    3   11   0    0    0   0   0    1    0
      DFFW   2    0    0   0    0    0   0   0    0    1
      DFMF   1    0    0   0    0    0   0   0    0    0
      FW     0    0    0  60    0   11   0   0    0    2
      FWDF   0    0    0   0    0    0   0   0    0    0
      FWMF   0    1    0  16    2   38   0   3    1   14
      GK     0    0    0   0    0    0  36   0    0    0
      MF     0    0    2   0    0    4   0 118    9   13
      MFDF   0    0    0   0    0    0   0   1    0    0
      MFFW   0    2    0   3    2   16   0   7    2   22

Overall Statistics

               Accuracy : 0.7862
                 95% CI : (0.7514, 0.8181)
    No Information Rate : 0.3405
    P-Value [Acc > NIR] : < 2.2e-16
```

Figura 5: Matriz de confusión del clasificador entrenado con Grandient Boosted Machine.

```
          Reference
Prediction  DF DFFW DFMF  FW FWDF FWMF  GK  MF MFDF MFFW
      DF   206    3   12   0    0    0   0   1    1    0
      DFFW   0    0    0   0    0    0   0   0    0    0
      DFMF   0    0    0   0    0    0   0   0    0    0
      FW     0    0    0  61    0   17   0   0    0    1
      FWDF   0    0    0   0    0    0   0   0    0    0
      FWMF   0    2    0  16    4   36   0   2    1   18
      GK     0    0    0   0    0    0  36   0    0    0
      MF     1    0    1   0    0    1   0 115   11   10
      MFDF   0    0    0   0    0    0   0   0    0    0
      MFFW   0    1    0   2    0   15   0  11    0   23

Overall Statistics

               Accuracy : 0.7845
                 95% CI : (0.7497, 0.8166)
    No Information Rate : 0.3405
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7252
```

Figura 6: Matriz de confusión del clasificador entrenado con Support Vector Machine.

Para analizar los resultados, aunque todas comparten la misma estructura y valores similares debido al parecido rango de precisión entre los clasificadores, utilizaremos la matriz relativa al modelo entrenado con *Random Forest*.

En primer lugar, a primera vista se puede observar en el eje de cada matriz todas las posiciones de los jugadores con las que se han trabajado en el problema. Esta forma de mostrar los resultados nos ayuda a comprobar respecto a cada posición original de un jugador, dónde ha sido finalmente clasificado a partir del modelo entrenado en apartados anteriores, dependiendo de las características que refleje el jugador en cuanto al rendimiento mostrado en todos los parámetros que registra el conjunto de datos.

En este tipo de problema, la clasificación que realiza la matriz de confusión es muy útil, ya que debido a la similitud que puede existir entre una posición pura (Defensa) y otra derivada de la misma (Defensa-mediocentro) en las estadísticas reflejadas por sus respectivos jugadores debido a las acciones que desempeñan en el terreno de juego, puede que se dé el caso de que, aunque una predicción no haya sido acertada, tenga cierto grado de relación y sentido la posición errónea que ha sido predicha por parte del clasificador. Se repasará posición por posición lo mostrado en la primera matriz:

Anotación: las posiciones que tengan una fila de ceros, representan que en la división del conjunto de test no se ha llegado a contener ningún registro del jugador con esa posición, por lo tanto, no se ha llegado a clasificar ningún jugador en dichas posiciones. Esto se debe a que suelen ser combinaciones de posiciones muy poco comunes en el conjunto de datos, por lo que es posible que en la división entre entrenamiento/test, se haya dejado fuera del conjunto jugadores con determinadas posiciones.

- Para los **defensas**, se han clasificado 207 correctamente como defensas puros. El resto de clasificaciones, no han sido totalmente acertadas, pero desde un punto de vista futbolístico, pueden llegar a ser posibles debido a la similitud entre las tareas que desempeñan, por lo que el clasificador no se ha alejado demasiado de la clasificación total de defensa: esto se debe a que el resto de defensas han sido clasificados en menor medida como defensas-delantero (3) y defensas-mediocentros (12).

- Para los **delanteros**, se han clasificado 62 jugadores correctamente como delanteros puros. El resto de clasificaciones interpretadas como erróneas por la matriz de confusión, sigue teniendo una misma lógica como la presentada anteriormente, ya que el resto de delanteros puros se han clasificado como delanteros-mediocentros, una posición derivada que desempeñan tareas muy similares por lo que es normal y totalmente posible que compartan estadísticas parecidas y que el clasificador lo haya interpretado de dicha forma.

- Para los **mediocentros**, se han clasificado 118 correctamente como mediocentros puros. Esta posición es la que la matriz de confusión muestra que el clasificador ha realizado más predicciones erróneas, pero, sin embargo, continúan teniendo cierta lógica. La posición de mediocentro es una

de las más flexibles en el fútbol, pudiendo ejercer tanto tareas defensivas como ofensivas al mismo tiempo, sin tener que ejercer un principal enfoque en las mismas. Esto se demuestra con la actual proporción de predicciones, pues los jugadores mediocentros que no han sido clasificados como los mismos, se les ha otorgado clases derivadas de la posición de mediocentro, como puede ser mediocentro-delantero o medio-centro defensa.

- Para los **porteros** se han clasificado 36 correctamente. Esta es la posición más fácil de predecir, sin error alguno, debido a la gran diferencia que hay entre las estadísticas que registra un portero en comparación con el resto de posiciones.
- Finalmente, para el resto de clases derivadas de las posiciones principales, se han obtenido de forma general unos resultados bastante precisos, pero más dispersos que en el caso de las posiciones puras, esto se debe a que las posiciones derivadas mezclan la lógica y el rendimiento de dos posiciones, por lo que resulta más difícil de clasificar exactamente, por ejemplo, un mediocentro-delantero como mediocentro-delantero. En este caso, es normal que se clasifique como otro tipo de mediocentro o que se le otorgue la etiqueta de delantero puro, nuevamente debido a los roles desempeñados y a las estadísticas que registre a consecuencia de estos.

4. Aprendizaje no supervisado

Esta parte está desarrollada por **Vincenzo Parretta** y **Félix Jiménez González**. Hablaremos de dos partes principales:

- Clustering
- Reglas de asociación

4.1. Clustering

En esta sección se desarrollará el uso de clustering para el análisis de los datos de estadísticas de desempeño de jugadores de fútbol (i.e. nuestro dataset de trabajo).

4.1.1. Paquetes utilizados

Más allá de los paquetes utilizados en el aprendizaje supervisado, se han utilizado:

- **stats** Utilizado para funciones estadísticas.
- **cluster** Utilizado para métodos de análisis de clústers.
- **corrplot** Utilizado para mostrar la matriz de correlación.
- **gridExtra** Utilizado para trabajar con gráficas de visualización.
- **GGally** Utilizado, nuevamente, para visualizar mejor la información.
- **knitr** Utilizado para extraer informes de resultados.

4.1.2. Preprocesamiento

Para el preprocesado de los datos, además de lo realizado en la sección de aprendizaje no supervisado, se ha realizado un escalado previo a la aplicación del clustering.

4.1.3. Elección del número de clústers

Para realizar el clustering, el primer paso que se debe dar es la elección del número de clústers que utilizar en el método k-means. Para hacer esta elección, se procede a analizar la compactación del dataset basándose en el número de clústers.

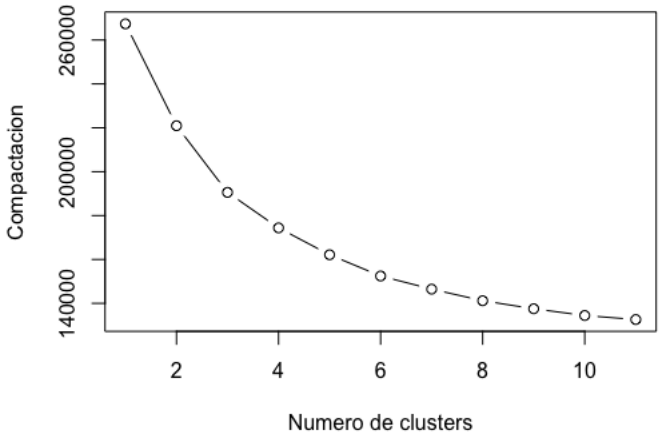


Figura 7: Número de clústers en base a la compactación.

Podemos ver que existe un cambio substancial de valores entre el primer y segundo clúster, por lo que se puede llegar a pensar que debemos utilizar dos clústers.

Sin embargo, este es un método de los muchos que hay para elegir el número óptimo de clústers. Ejecutando estos otros métodos se puede ver que el número óptimo de clústers es tres.

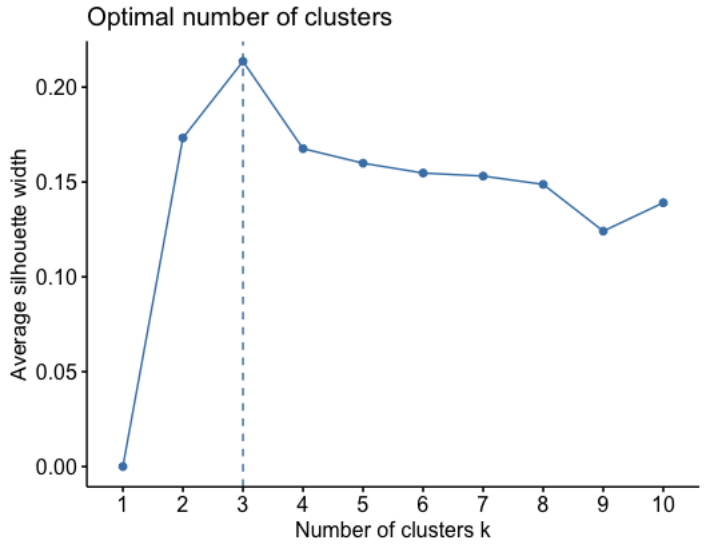


Figura 8: Número óptimo de clústers utilizando el método Silhoutte.

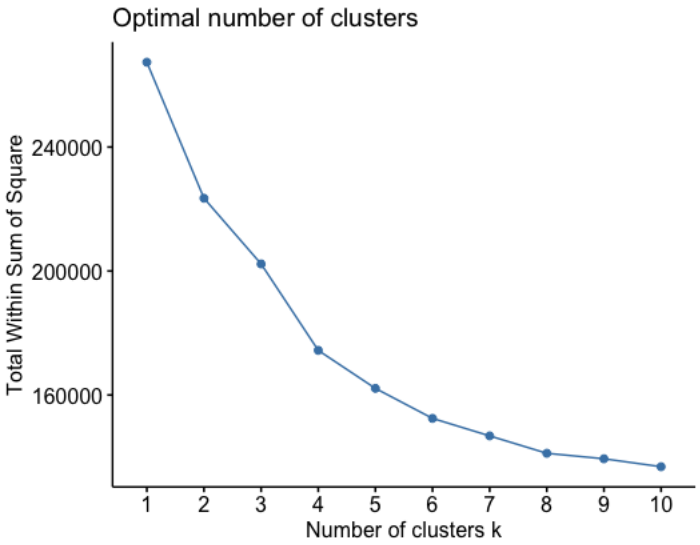


Figura 9: Número óptimo de clústers utilizando el método WSS.

4.1.4. K-Means y resultados obtenidos

Una vez tenemos el número óptimo de clústers, utilizamos el método K-Means y hemos creado varias gráficas para visualizar los resultados del clustering.

Para empezar, realizamos el método k-means con las estadísticas de goles/tiros que han realizado los jugadores, obteniendo los siguientes resultados:

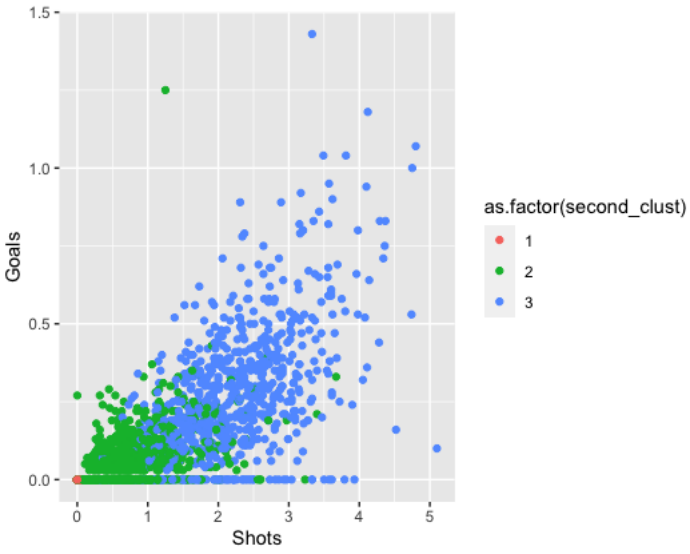


Figura 10: 3 clústers con goals/shots.

Sin embargo, se hizo en clasificación un estudio de las variables más importantes del dataset, obteniendo así que estas eran **TouDef3rd** y **PresMid3rd**.

Por lo que aplicando K-Means a estas variables, tenemos los siguientes resultados:

Figura 11: 3 clústers con TouDef3rd/PresMid3rd.

4.1.5. Clustering jerárquico

Para proceder con el clustering jerárquico, se deben seguir los siguientes pasos:

1. El primer paso consiste en crear la matriz de distancias. Se pueden utilizar varios tipos de distancias(distancia euclídea, distancia Manhattan, etc.).

2. En el paso siguiente (y considerando cada objeto como un clúster), se combinan los dos clústeres más cercanos y se actualiza la matriz de distancias con esa información (repitiendo así el proceso hasta quedar únicamente un clúster).

Con la función *hclust*, se pueden especificar los distintos métodos de aglomeración que se pueden utilizar (*complete*, *single*, *average*, ...) así como las distancias disponibles (por ejemplo, las distancias Canberra, Manhattan, Euclídea, ...).

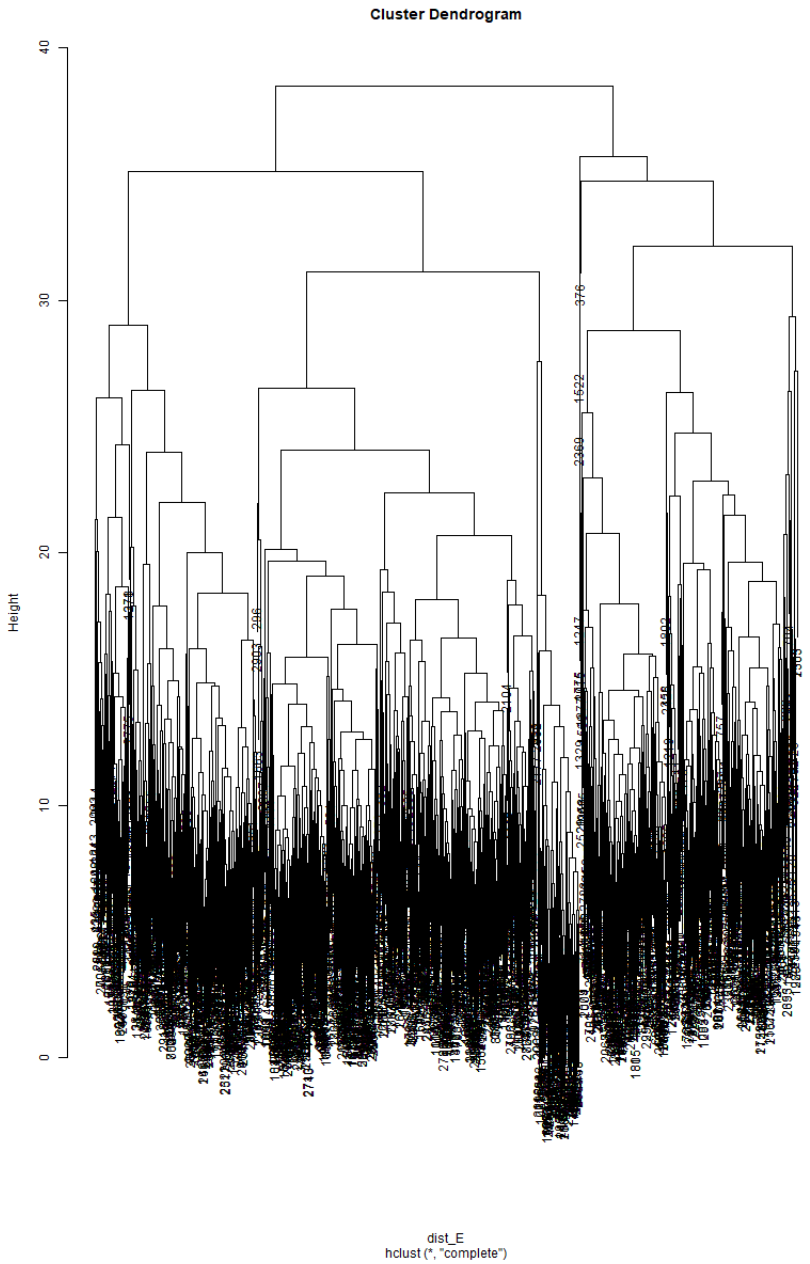


Figura 12: Dendrograma utilizando la distancia euclídea y el método completo.

Aparte de los métodos *single* y *complete*, se pueden utilizar métodos estadísticos para analizar el clúster (nuevamente la mala visibilidad del dendrograma se debe a la gran cantidad de columnas del dataset).

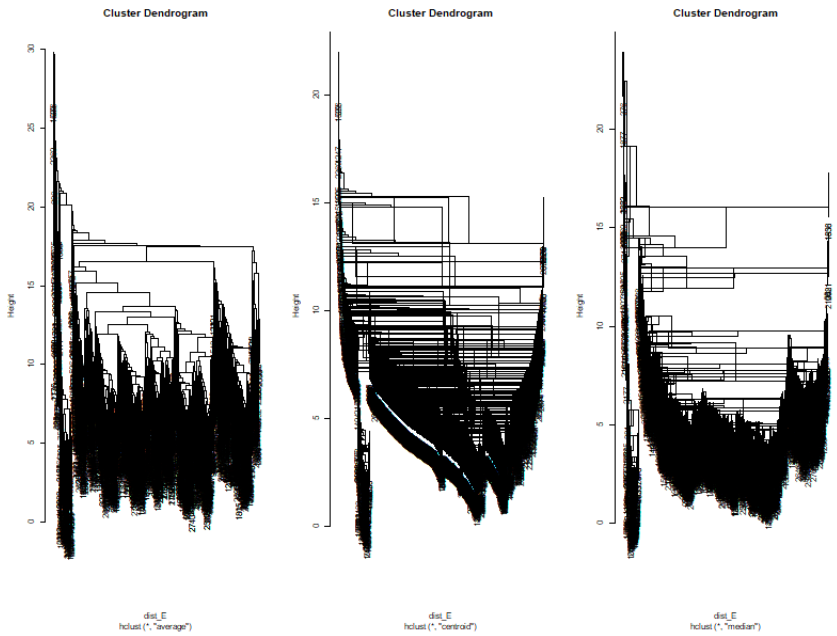


Figura 13: Dendrograma utilizando métodos estadísticos.

4.2. Reglas de asociación

En esta sección se procede a desarrollar para nuestro dataset, el uso de *reglas de asociación*.

El objetivo que se busca al aplicar reglas de asociación se basa en encontrar relaciones dentro de un conjunto de transacciones que tienden a ocurrir de manera conjunta.

Existen varios métodos diseñados para identificar reglas de asociación, de los cuales se pueden destacar:

- **Apriori**, el cual consta de dos etapas:
  1. Identificar los *itemsets*, o conjunto de eventos que ocurren de manera frecuente.
  2. Conversión de estos *itemsets* en reglas de asociación.
- **FP-Growth** que, a diferencia del método *Apriori*, los itemsets se identifican sin necesidad de generar candidatos para cada tamaño.

- **Eclat**, diferenciándose del método *Apriori* en la manera de analizar los datos.

4.2.1. Paquetes utilizados

Adicionalmente, a los paquetes mencionados en las secciones anteriores haremos uso de los paquetes:

- **arules**, **arulesViz**, que implementan el algoritmo *Apriori* para la identificación de itemsets frecuentes y la creación de reglas de asociación a través de la función *apriori()* y la visualización de los datos respectivamente.

4.2.2. Preprocesamiento

Realizamos el mismo preprocesamiento que el aplicado en clustering.

4.2.3. Análisis de ítems del dataset

Para empezar, se debe convertir el dataframe a un objeto de tipo *transacción*, i.e. una lista en la que cada elemento contiene los ítems de una transacción. Esto se consigue a través de la función *as* y pasándole la clase "transactions".

Una vez obtenidas las transacciones, mostramos de manera más visual la distribución de los tamaños de las transacciones a través de un histograma.

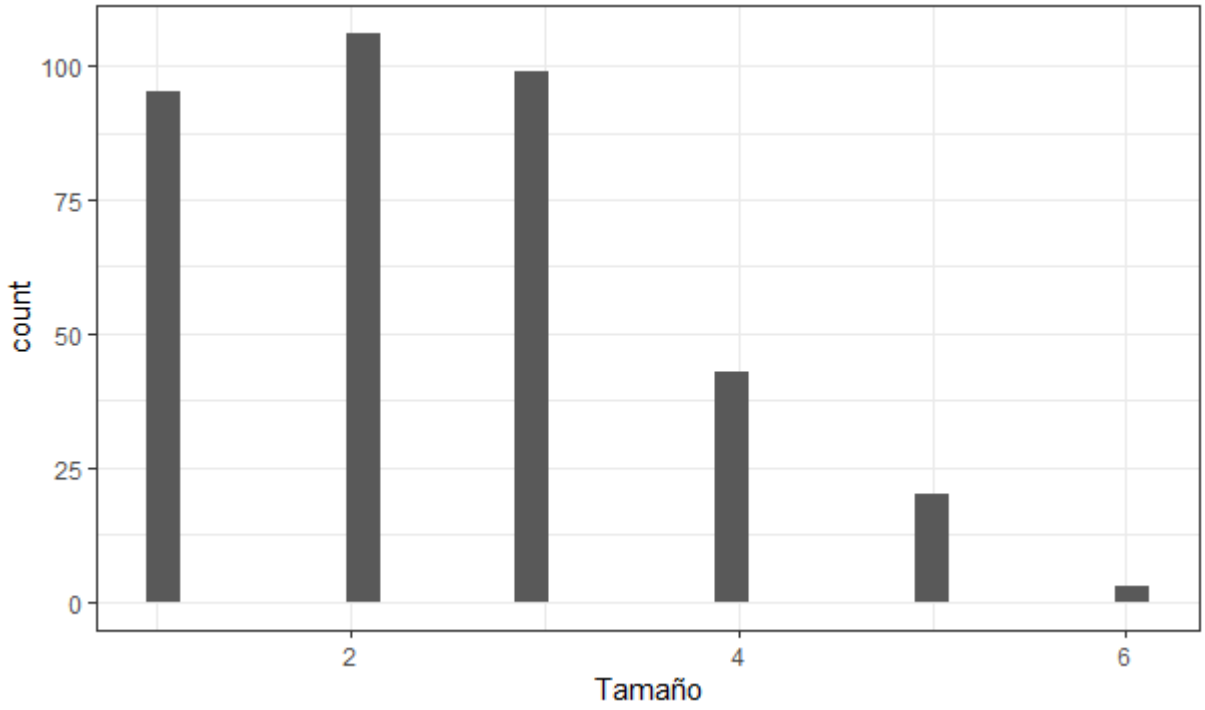


Figura 14: Distribución del tamaño de las transacciones.

4.2.4. Análisis de frecuencia

Otro objeto de estudio es la frecuencia de los distintos ítems que se han recuperado. Esto se consigue a través de la función itemFrequency(), la cual puede obtener la frecuencia relativa o la frecuencia absoluta (número de transacciones que aparece en cada ítem).

```
> frecuencia_items %>% sort(decreasing = TRUE) %>% head(10)
        MF         FW       FWMF         DF       MFFW       MFDF       DFMF
0.50819672 0.46174863 0.42076503 0.38524590 0.34972678 0.10655738 0.10109290
      DFFW       FWDF         GK
0.05464481 0.04371585 0.01092896
```

Figura 15: Frecuencia relativa.

```
> frecuencia_items %>% sort(decreasing = TRUE) %>% head(10)
  MF   FW FWMF   DF MFFW MFDF DFMF DFFW FWDF   GK
 186  169  154  141  128   39   37   20   16    4
```

Figura 16: Frecuencia absoluta.

4.2.5. Obtención de itemsets

La extracción de los itemsets viene de la mano de la función *apriori()*, que además puede obtener las reglas de asociación que superen un *support* de confianza concreto.

A continuación se muestran los datos devueltos por los *itemsets* conseguidos para nuestro dataset.

```
most frequent items:
     MF    MFFW      FW    FWMF      DF (Other)
     11       9       7       7       6       6

element (itemset/transaction) length distribution:sizes
 1  2  3
 7 12  5

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.000   2.000   1.917   2.000   3.000

summary of quality measures:
    support           count
 Min.   :0.08197   Min.   : 30.00
 1st Qu.:0.09290   1st Qu.: 34.00
 Median :0.13798   Median : 50.50
 Mean   :0.19695   Mean   : 72.08
 3rd Qu.:0.24590   3rd Qu.: 90.00
 Max.   :0.50820   Max.   :186.00
```

Figura 17: Itemsets con un support de 0.08196721.

Para visualizarlos mejor, pasamos, se puede pasar la información a un dataframe que mostramos con *ggplot*.

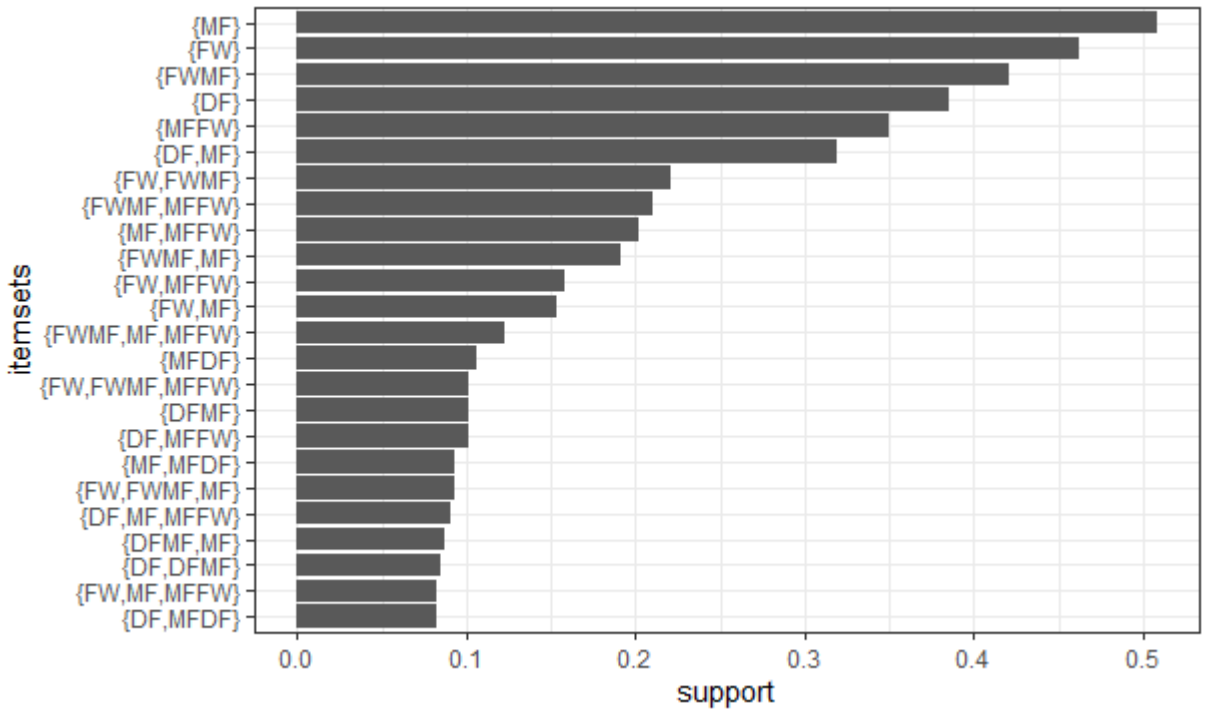


Figura 18: Itemsets más frecuentes.

4.2.6. Reglas de asociación

Para obtener las reglas de asociación en base a los datos capturados previamente, se debe pasar como parámetro a la función *apriori*, el *target* con value *rules*. Estas se pueden ordenar de diversas maneras. En nuestro caso, se ordernarán por *confidence*.

```
     lhs            rhs  support    confidence coverage  lift     count
[1] {DF, MFFW} => {MF} 0.09016393 0.8918919  0.1010929 1.755013  33
[2] {MFDF}     => {MF} 0.09289617 0.8717949  0.1065574 1.715467  34
[3] {DFMF}     => {MF} 0.08743169 0.8648649  0.1010929 1.701831  32
[4] {DFMF}     => {DF} 0.08469945 0.8378378  0.1010929 2.174813  31
[5] {DF}       => {MF} 0.31967213 0.8297872  0.3852459 1.632807 117
[6] {MFDF}     => {DF} 0.08196721 0.7692308  0.1065574 1.996727  30
```

Figura 19: Reglas obtenidas.

4.2.7. Evaluación de las reglas resultantes

La evaluación de las reglas se puede llevar a cabo gracias al método *interestMeasure*, capaz de calcular más de 20 métricas distintas. En nuestro caso, reflejaremos la evaluación utilizando el *coverage* y el *fishersExactTest*

```
  coverage fishersExactTest
1 0.1010929     3.456932e-09
2 0.1010929     2.064905e-06
3 0.1065574     2.894313e-07
4 0.1065574     5.897148e-07
5 0.3852459     1.396697e-23
6 0.1010929     2.647837e-07
```

Figura 20: Métricas usando coverage y fishersExactTest.

4.2.8. Filtrado de reglas

A la hora de crear las reglas de asociación, es posible que se consideren interesantes aquellas que contienen unos items determinados. Para obtener estas reglas, es posible recurrir al filtrado de las reglas de asociación. Existen diversas maneras de filtrar:

- Reglas maximales. Las reglas maximales son aquellas que se generan a partir de itemsets maximales. Los itemsets son maximales si no existe otro itemset que crea su superset. Para identificar las reglas maximales, se puede utilizar la función *is.maximal(reglas)*.

```
> reglas_maximales <- reglas[is.maximal(reglas)]
> reglas_maximales
set of 5 rules
> inspect(reglas_maximales)
    lhs           rhs  support    confidence coverage  lift     count coverage
[1] {DFMF}     => {DF} 0.08469945 0.8378378  0.1010929 2.174813 31    0.1010929
[2] {DFMF}     => {MF} 0.08743169 0.8648649  0.1010929 1.701831 32    0.1010929
[3] {MFDF}     => {DF} 0.08196721 0.7692308  0.1065574 1.996727 30    0.1065574
[4] {MFDF}     => {MF} 0.09289617 0.8717949  0.1065574 1.715467 34    0.1065574
[5] {DF, MFFW} => {MF} 0.09016393 0.8918919  0.1010929 1.755013 33    0.1010929
    fishersExactTest
[1] 3.456932e-09
[2] 2.064905e-06
[3] 2.894313e-07
[4] 5.897148e-07
[5] 2.647837e-07
```

Figura 21: Reglas maximales resultantes.

- Reglas redundantes. Se consideran redundantes aquellas reglas cuyo consecuente puede obtenerse a partir de reglas conocidas. Para identificar aquellas reglas que son redundantes, podemos hacer uso de la función is.redundant(reglas).

```
> reglas_redundantes <- reglas[is.redundant(x = reglas)]
> reglas_redundantes
set of 0 rules
```

Figura 22: Reglas redundantes resultantes.

- Transacciones que verifican una determinada regla. Una vez tenemos una regla en concreto, se pueden filtrar las transacciones en las que se cumple. Un ejemplo puede ser la recuperación de aquellas transacciones para las que se cumple la regla con mayor confianza:

```
[1] "{DF,MFFW} => {MF}"
```

Figura 23: Transactions.

  En nuestro caso, las transacciones que cumplen esta regla son todas aquellas que contienen DF, MFFW y MF.

  Se procede a mostrar un subset de 3 de estas transacciones:

```
> inspect(filtrado_transacciones[1:3])
    items                          transactionID
[1] {DF, DFMF, FWDF, MF, MFFW} 0.32
[2] {DF, MF, MFFW}             0.5
[3] {DF, MF, MFFW}             0.64
```

Figura 24: Filtrado de transacciones.

5. Estudio de la herramienta BIGML

En esta sección se muestra el análisis de un conjunto de datos proporcionado por la plataforma BIGML; en particular, está el estudio de la técnica de predicción.

5.1. Elección de la fuente, el dataset y la generación del modelo

Después de acceder a la plataforma, puede ver la página **Fuentes** que muestra todas las fuentes de datos cargadas por defecto en la propia plataforma. Se eligió trabajar en el archivo **Titanic Survival** que contiene campos como:

- **Age** indica la edad de los pasajeros.
- **ClassDept** indica la clase a la que cada pasajero pertenece.
- **FareToday** indica el precio del billete.
- **Joined** indica el puerto de embarque
- **Survived** indica la salida de la predicción del modelo en cuestión.

La predicción que vamos a hacer se basa en la posibilidad de supervivencia de un pasajero genérico en función de las características y datos presentes en los campos mencionados anteriormente.
En este punto, es posible crear el conjunto de datos en el que se está trabajando y construir el modelo; además, con esta operación si se generan histogramas que muestran todas las ocurrencias de los distintos campos: este no es nuestro caso, pero si se requiere un histograma completamente plano, tendemos a destapar estos campos para que todas las ocurrencias sean constantes inútiles para el análisis que íbamos a hacer. El panel de control se presenta así:

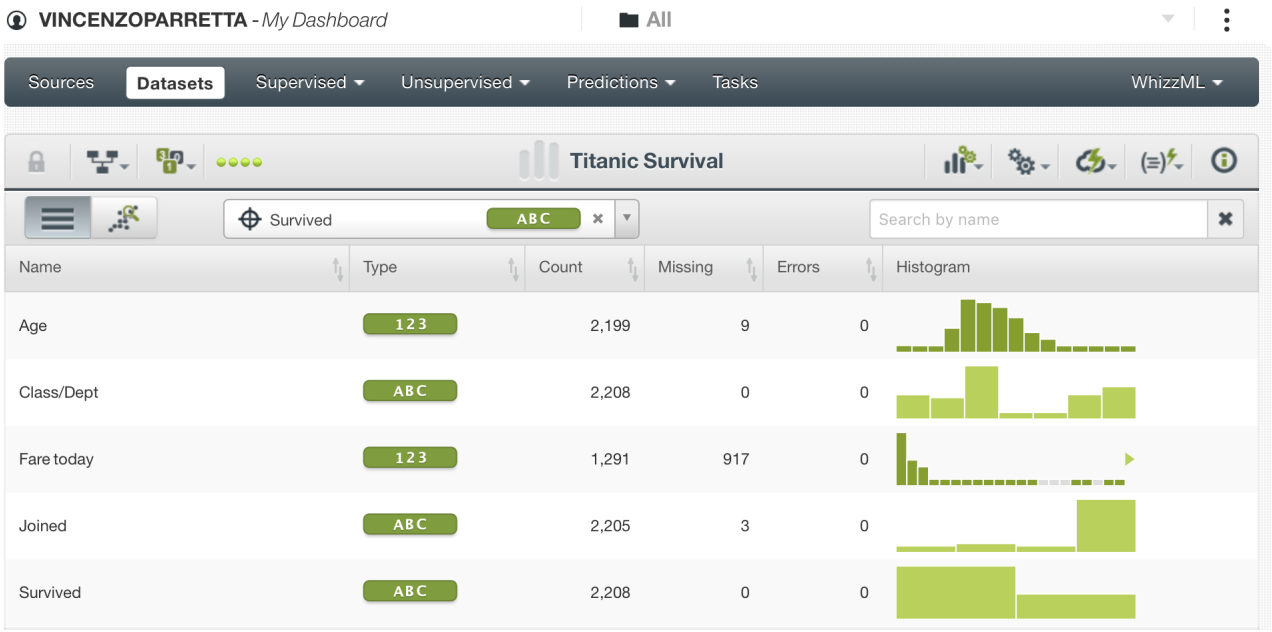


Figura 25: Presentación de dataset y histogramas

En cuanto a la construcción del modelo, es necesario decir que a través de este es posible generar pronósticos para los valores que puede asumir el campo "Target/Objective Field". El modelo que creamos se basa en un algoritmo de aprendizaje automático basado en aprendizaje supervisado y, se genera el siguiente árbol de decisión:

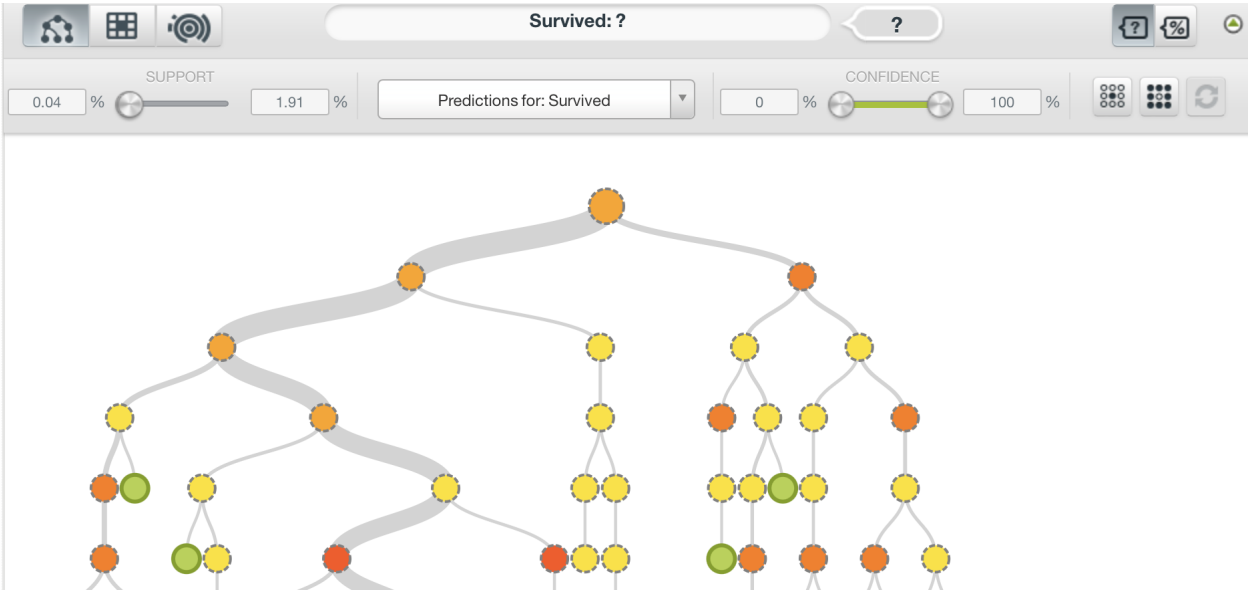


Figura 26: Árbol de decisión

Con este árbol es posible ver cómo cada ruta desde un nodo padre a un nodo hijo representa una combinación de campos que determina si los pasajeros pueden sobrevivir o no. Por ejemplo, como se muestra en la siguiente figura, se puede observar que para ese nodo seleccionado la probabilidad de no supervivencia de un pasajero mayor de 27 años que pertenezca a la clase definida como "a la carta" es igual al 83,89 por ciento y se basa en un número de 20 instancias.

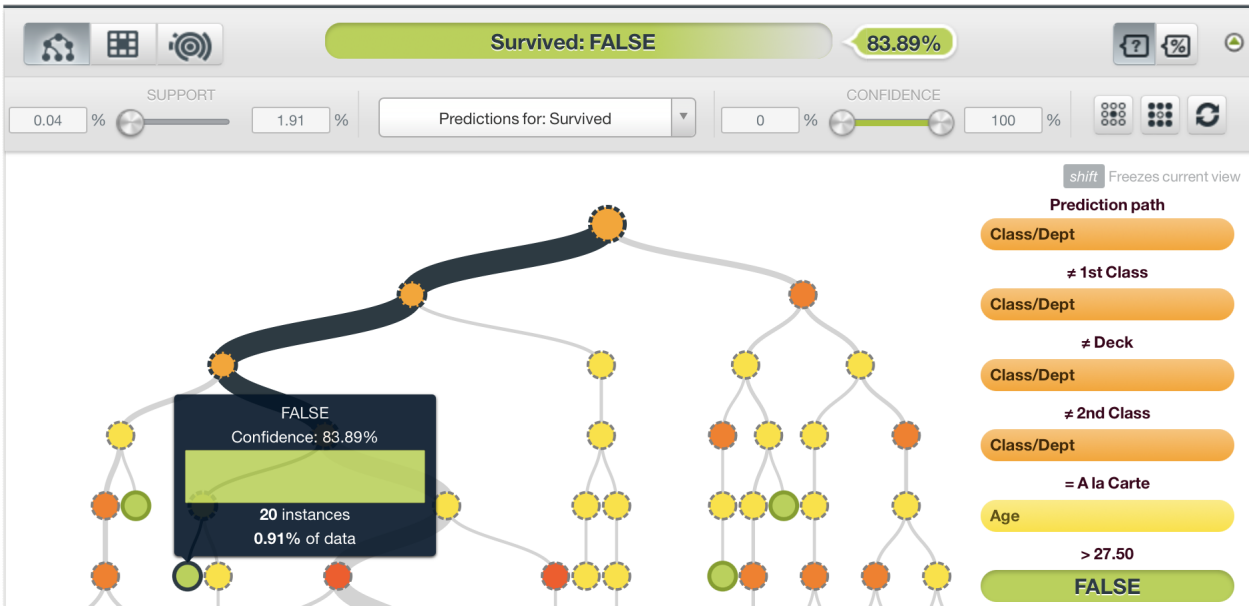


Figura 27: Árbol de decisión con prediction path

5.2. Predicción

Con la herramienta bigML es posible tener tres posibilidades de predicción:

- **Prediction by questions** con respuesta a preguntas propuestas por el mismo modelo y predicción de una sola instancia.
- **Predict** con una máscara con campos de datos de entrada y predicción de instancia única.
- **Batch Prediction** que le permite predecir múltiples instancias y obtenemos una predicción general.

En el **primer caso** es posible notar cómo para una respuesta a las preguntas en las que tenemos un pasajero de tercera clase, de 76 años y embarcado en Cherburgo, la probabilidad de supervivencia es falsa y con una confianza igual al 74,24 por ciento.

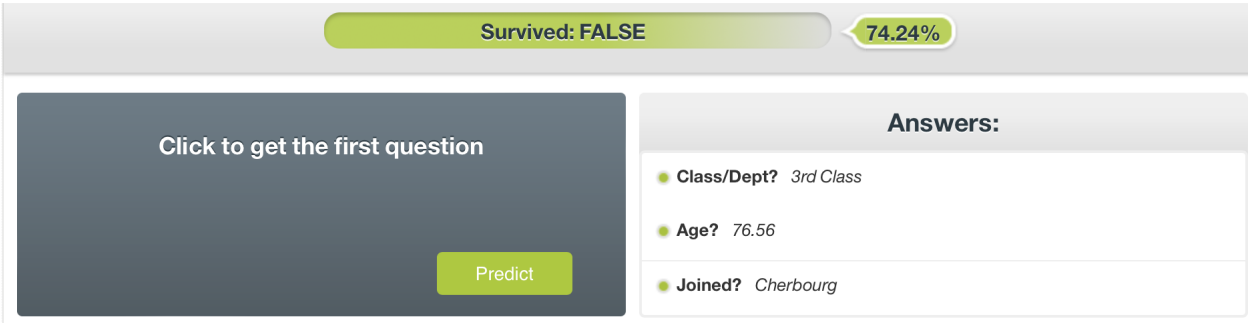


Figura 28: Ejemplo de "Prediction by questions"

Con el comando **Predict** tenemos la siguiente pantalla donde es posible configurar los valores de los campos de relevancia; es una herramienta muy rápida para ver cómo, al variar determinados campos, la probabilidad de supervivencia puede pasar de verdadera a falsa y viceversa. En este caso de ejemplo, se puede ver que para un pasajero de primera clase de 18 años, la probabilidad de supervivencia es aproximadamente igual al 80 por ciento.

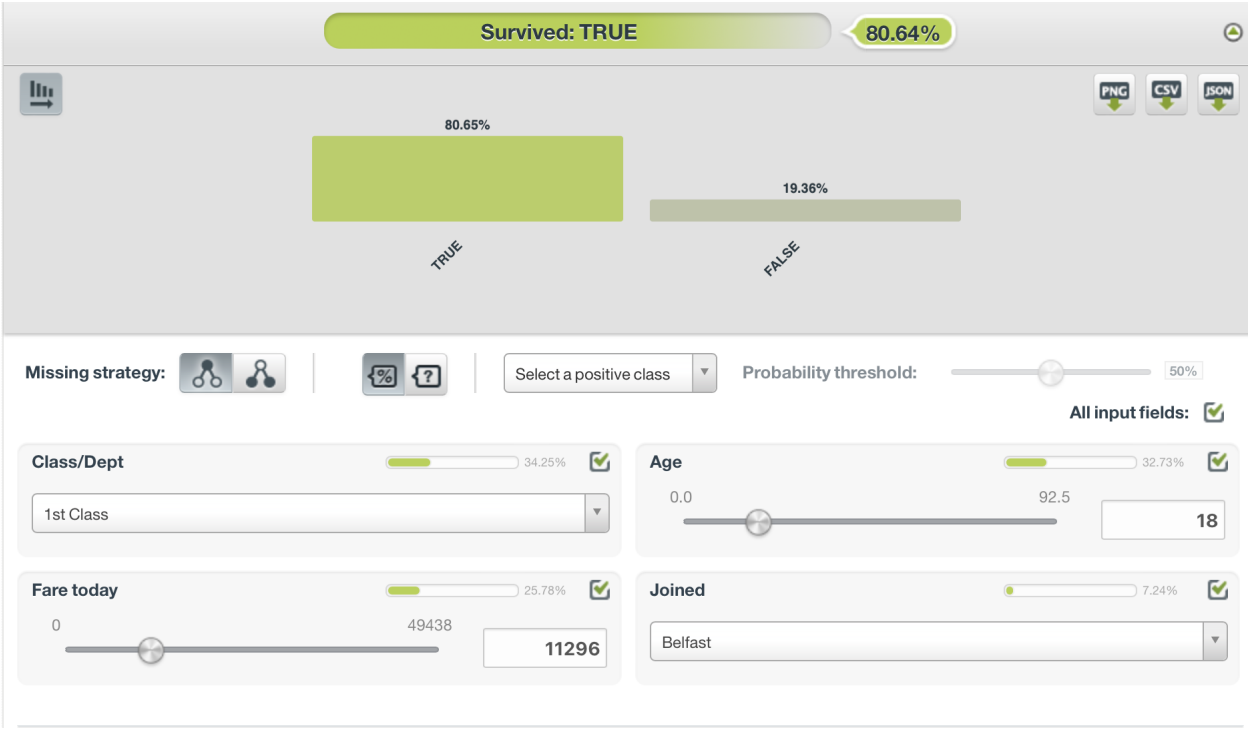


Figura 29: Ejemplo de "Prediction"

Finalmente, como se escribió anteriormente, es posible tener una **Batch Prediction** que permita proporcionar un conjunto de datos de ocurrencias de entrada y obtener una predicción de salida para

cada una de estas. En particular, para hacer esta predicción es importante notar cómo, nuevamente utilizando la herramienta bigML, fue posible dividir el conjunto de datos inicial en dos conjuntos de datos: uno para entrenamiento e igual al 80 por ciento del conjunto de datos original y otro para la prueba igual. al 20 por ciento del conjunto de datos inicial. Luego, usamos el conjunto de datos de prueba y generamos el pronóstico que no es más que un nuevo conjunto de datos en el que es posible mostrar el campo esperado y el realmente pronosticado.

A	B	C	D	E	F	G
Age	Class/Dept	Fare today	Joined	Survived?	Survived	Match?
18.0	1st Class	19100	Cherbourg	TRUE	TRUE	VERO
60.0	1st Class	2000	Cherbourg	FALSE	FALSE	VERO
24.0	1st Class	19100	Cherbourg	FALSE	TRUE	FALSO
30.0	1st Class	12700	Southamptor	TRUE	TRUE	VERO
61.0	1st Class	2050	Southamptor	TRUE	FALSE	FALSO
45.0	1st Class	20300	Cherbourg	TRUE	TRUE	VERO
41.0	1st Class	2360	Southamptor	FALSE	FALSE	VERO
48.0	1st Class	3900	Cherbourg	FALSE	FALSE	VERO

Figura 30: Ejemplo de "Batch Prediction"

Con ello se analizó el porcentaje de acierto de los resultados obtenidos: la predicción fue igual al valor real en el 81 por ciento de los casos.

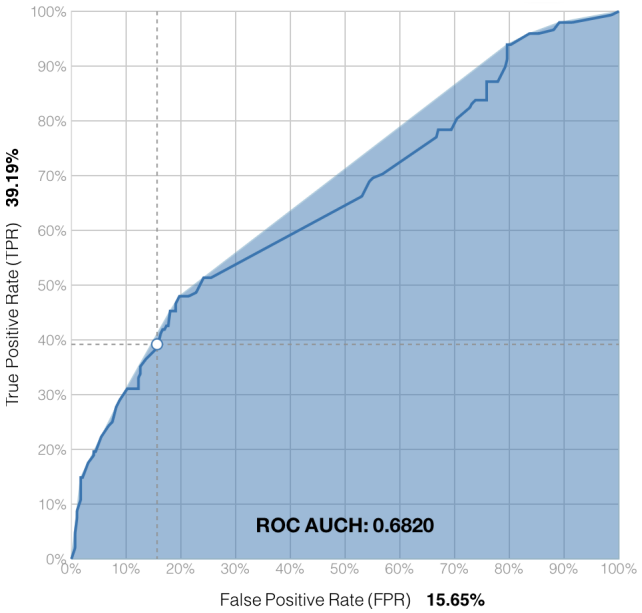
5.3. Evaluación del modelo de predicción

Para evaluar el modelo de predicción generado, vaya a la sección **Evalutation**, nuevamente proporcionada por la herramienta bigML. En particular, es necesario insertar, utilizando la herramienta, tanto el conjunto de datos de entrenamiento como el conjunto de datos de prueba generado aleatoriamente previamente. Una vez que se evalúa el modelo, bigML crea una representación de todas las métricas necesarias para la evaluación en sí. Por ejemplo, es posible ver cómo se genera una matriz de confusión en la que tenemos los valores reales en las columnas y los valores reales generados por la predicción en las filas. En nuestro caso, tenemos un número de TF igual a 248 y un número de TP igual a 58 mientras que hay un número de FP igual a 90 y un número de FN igual a 46.

ACTUAL VS. PREDICTED	TRUE	FALSE	ACTUAL	RECALL	F	Phi
TRUE	58	90	148	39.19%	0.46	0.26
FALSE	46	248	294	84.35%	0.78	0.26
PREDICTED	104	338	442	61.77% AVG. RECALL	0.62 AVG. F	0.26 AVG. Phi
PRECISION	55.77%	73.37%	64.57% AVG. PRECISION	69.23% ACCURACY		

Figura 31: Matriz de confusión

Puede ver cómo la herramienta también proporciona todas las métricas, como exactitud, precisión, recuperación, F-Measure y coeficiente Phi. También es posible generar una curva ROC en la que esté el TPR en el eje de ordenadas y el FPR en el eje de abscisas.



69.23% Accuracy		0.4603 F-measure
55.77% Precision	39.19% Recall	0.2619 Phi coefficient
15.65% FPR	23.53% % positive instances	166.55% Lift
28.24% K-S statistic	0.2103 Kendall's Tau	0.2497 Spearman's Rho

Figura 32: Curva ROC y métricas