Treball de Fi de Grau/Màster

Màster Universitari en Enginyeria d'Organització (MUEO)

Comparación de algoritmos de clasificación supervisada

MEMÒRIA

Autor: Alexandre Serra Marrugat
Director: Josep Ginebra

Convocatòria: Juliol 2020

Escola Tècnica Superior

d'Enginyeria Industrial de Barcelona

RESUMEN

En este trabajo se comparan los principales algoritmos de clasificación de aprendizaje supervisado basados en árboles de decisión. Los algoritmos que se utilizarán en el estudio son: árbol de decisión, bagging, Random Forest y AdaBoost. El objetivo es observar cuál de estos algoritmos se comporta mejor.

Para entender que son los algoritmos de clasificación de aprendizajes supervisados primero se detallará una ampliación general del aprendizaje automáticos y sus posibles clasificaciones y aplicaciones.

En una primera parte del trabajo se definen cada uno de los algoritmos de manera teórica, consiguiendo así, entender de manera detallada su comportamiento, sus características, ventajas y desventajas.

La segunda parte del trabajo consistirá en aplicar cada uno de estos algoritmos a un conjunto de datos para observar cuáles son sus prestaciones. Se emplearán un conjunto de datos que hacen referencia a múltiples reservas de hoteles. El objetivo del modelo será predecir si una reserva será cancelada o no. Para ello se usará Python, y en concreto, su librería de aprendizaje supervisado Scikit-Learn.

Previo al uso de los algoritmos, el conjunto de datos ha sido estudiado con atención para detectar si había algunos datos erróneos o atributos que no eran útiles para nuestra aplicación del aprendizaje supervisado.

Para conseguir el mejor modelo de cada uno de los algoritmos se han realizado dos pasos: seleccionar los atributos más importantes y ajustar los hiperparámetros. De esta manera conseguimos que los modelos obtengan unas prestaciones muchos mejores. Para realizar estos entrenamientos y ajustes se ha dividido el conjunto de datos en datos de entrenamiento, datos de validación y datos de test. De este modo se evita en lo posible presentar resultados sobre ajustados.

Tras entrenar y ajustar cada uno de los modelos, se ha podido observar que el modelo que mejor se comporta con nuestro conjunto de datos es el Random Forest, seguido muy de cerca del bagging.

CONTENIDO

1 INTRODUCCIÓN

El trabajo se basa en aprendizaje automático, es decir, en que las maquinas aprenden por sí solas para conseguir modelos útiles. Un tipo de aprendizaje automático es el aprendizaje supervisado de clasificación, que consiste en clasificar en categoría a partir de información previa de datos clasificados correctamente.

Dentro de esta categoría de aprendizajes hay múltiples algoritmos. Nos centraremos en los más importantes, en los que destacan los algoritmos de árboles de decisión y Random Forest. Para entender bien los modelos, se hará tanto un estudio teórico como un estudio practico con el uso de un conjunto de datos.

1.1 OBJETIVOS DEL PROYECTO

Los objetivos que se han planteado para ser alcanzados durante la realización de este proyecto han sido:

- Entender que es el aprendizaje automático.
- Comparar y entender teóricamente los algoritmos de clasificación: árboles de decisión, bagging, Random Forest y AdaBoost.
- Limpiar y entender un conjunto de datos real.
- Entrenar correctamente los modelos de aprendizaje supervisados mencionados.
- Trabajar con los algoritmos de manera práctica usando programación Python y la librería Scikit-Learn.
- Extraer las prestaciones y resultados de los modelos entrenados.
- Comparar los resultados prácticos de los algoritmos.

1.2 ALCANCE DEL PROYECTO

Este proyecto se ha definido con la intención de solo mostrar algunos de los algoritmos y técnicas del aprendizaje supervisado de clasificación. Hay otros tipos de aprendizaje automático, como por ejemplo el no supervisado o el reforzado, pero no nos hemos centrado en estos.

Dentro de los algoritmos supervisados de clasificación también hay muchos modelos a utilizar, pero en este trabajo solo se han estudiado cuatro (árboles de decisión, bagging, Random Forest y AdaBoost), siendo todos ellos, de la familia de los árboles de decisión.

En relación a la parte práctica, se ha usado solo un conjunto de datos para estudiar los diferentes comportamientos de los algoritmos. Las conclusiones del mejor modelo estarán únicamente a un conjunto de datos.

2 APRENDIZAJE AUTOMÁTICO

Actualmente vivimos en una "era de datos", donde se recolecta y almacena una gran cantidad de datos todos los días. Ante esta creciente cantidad de datos, los métodos de aprendizaje automático han tomado una gran importancia. Es muy probable que una persona use docenas de veces estos métodos sin ni siquiera darse cuenta.

Un ejemplo de aprendizaje automático "cotidiano" para millones de usuarios es el algoritmo que está detrás del generador de noticias de Facebook. Facebook utiliza el aprendizaje automático para explotar los datos y comentarios de los usuarios para personalizar sus noticias. Si le gusta una publicación o deja de desplazarse para leer algo, el algoritmo aprende de esto y comienza a llenar sus noticias con contenido similar. Este aprendizaje se realiza de forma continua, por lo que el material sugerido en sus nuevas noticias evoluciona con sus preferencias, haciendo que su experiencia de usuario sea más agradable.

Hay muchos otros ejemplos, Apple reconoce la cara de tu amigo en una foto que acabas de hacer. Alexa de Amazon puede comprender y puede responder a preguntas. Netflix recomienda constantemente videos que coinciden con tus gustos. El aprendizaje automático se ha convertido en una parte muy importante de nuestra vida diaria.

Pero, ¿qué es exactamente el aprendizaje automático? ¿Qué hay detrás de estos famosos algoritmos? ¿Y cómo usan los datos para funcionar tan bien?

Formalmente, el aprendizaje automático es la ciencia que hace que las computadoras realicen una acción sin ser programadas explícitamente. En otras palabras, la gran diferencia entre los algoritmos de aprendizaje clásico y automático radica en la forma en que los definimos.

Los algoritmos clásicos reciben reglas exactas y completas para completar una tarea. En cambio, los algoritmos de aprendizaje automático reciben pautas generales que definen el modelo, junto con los datos. Estos datos deben contener la información faltante necesaria para que el modelo complete la tarea. Por lo tanto, un algoritmo de aprendizaje automático puede cumplir su tarea cuando el modelo se ha ajustado con respecto a los datos. Por eso se dice que "ajustamos el modelo con los datos" o que "el modelo tiene que ser entrenado en los datos".

Es decir, los algoritmos clásicos, combinan reglas creadas por humanos con datos para crear respuestas a un problema. En cambio, el aprendizaje automático utiliza datos y respuestas para descubrir y aprender las reglas detrás de un problema.

Para aprender estas reglas, las máquinas tienen que pasar por un proceso de aprendizaje, probar diferentes reglas y aprender de qué tan bien funcionan. Es por este motivo, que se conoce como aprendizaje automático o en ingles machine learning.

En general, el aprendizaje automático es increíblemente útil para tareas difíciles cuando tenemos información incompleta o información que es demasiado compleja para codificarla a mano. En estos casos, podemos proporcionar la información que tenemos disponible para nuestro modelo y dejar que este "aprenda" la información que falta por sí misma. El algoritmo luego utilizará técnicas estadísticas para extraer el conocimiento faltante directamente de los datos.

3 TIPOS APRENDIZAJE AUTOMÁTICO

Hay muchas clasificaciones que se pueden hacer con los algoritmos de aprendizaje automático. Por lo general, se agrupan en las áreas que se enumeran a continuación. Supervisados y no supervisados son los más utilizados. El aprendizaje semi supervisado y de refuerzo son más nuevos y más complejos, pero recientemente han mostrado unos resultados muy buenos.

- Aprendizaje supervisado
- Aprendizaje no supervisado
- Aprendizaje semi supervisado
- Aprendizaje reforzado

El teorema de "No Free Lunch" es famoso en el aprendizaje automático. Establece que no existe un algoritmo único que funcione bien para cualquier problema. Cada tarea que se intenta resolver tiene sus propias características. Por lo tanto, hay muchos algoritmos y enfoques para adaptarse a las peculiaridades individuales de cada problema. Cada vez se irán introduciendo nuevos modelos o técnicas de aprendizaje automático e IA que se adapten mejor a diferentes problemas.

3.1 APRENDIZAJE SUPERVISADO

La mayoría de aprendizajes automático utilizan el aprendizaje supervisado.

El aprendizaje supervisado se da cuando se tiene variables de entrada (x) y una variable de salida (Y) y se utiliza un algoritmo para aprender la función de mapeo (las normas) de la entrada a la salida.

$$Y = f(X)$$

El objetivo es aproximar la función de mapeo tan bien que cuando se tengan nuevos datos de entrada (x) se pueda predecir la variable de salida (Y) para esos datos.

Se llama aprendizaje supervisado porque el proceso de aprender de un algoritmo de unos datos puede ser visto como que un profesor está supervisando el proceso. Se conocen las respuestas correctas, así que el algoritmo va realizando predicciones de manera iterativa con los datos que tiene y, estas predicciones, son corregidas por el profesor. El aprendizaje termina cuando el algoritmo consigue obtener unos resultados aceptables.

Los problemas de aprendizaje supervisados pueden agruparse en problemas de regresión y clasificación.

- Clasificación: Un problema de clasificación es cuando la variable de salida es una categoría, como "rojo" o "azul" o "enfermedad" y "sin enfermedad".
- Regresión: Un problema de regresión es cuando la variable de salida es un valor real, como "dólares" o "peso".

Algunos problemas típicos sobre la clasificación y la regresión podrían ser la recomendación y la predicción de series temporales respectivamente.

Algunos ejemplos populares de algoritmos supervisados de aprendizaje automático son:

- Regresión lineal para problemas de regresión.
- Bosque aleatorio (Random Forest) para problemas de clasificación y regresión.
- Soporte de máquinas de vectores (Support Vector Machine) para problemas de clasificación.

3.2 APRENDIZAJE NO SUPERVISADO

El aprendizaje no supervisado es donde solo se tiene datos de entrada (X) y no hay variables de salida correspondientes.

El objetivo del aprendizaje no supervisado es modelar la estructura o distribución subyacente en los datos para aprender más sobre los datos.

Un ejemplo de aprendizaje no supervisado en la vida real sería clasificar monedas de diferentes colores en pilas separadas. Nadie le enseñó cómo separarlos, pero solo mirando sus características como el color, puede ver qué monedas de color están relacionadas y agruparlas en sus grupos correctos.

Se denominan aprendizaje no supervisado porque, a diferencia del aprendizaje supervisado anterior, no hay respuestas correctas y no hay un maestro. Los algoritmos se dejan a sus propios dispositivos para descubrir y presentar la estructura interesante en los datos.

Los problemas de aprendizaje no supervisados pueden agruparse en problemas de agrupamiento y asociación.

- Agrupación: un problema de agrupación es donde desea descubrir las agrupaciones inherentes en los datos, como agrupar clientes por comportamiento de compra.
- Asociación: un problema de aprendizaje de reglas de asociación es cuando desea descubrir reglas que describen grandes porciones de sus datos, como las personas que compran X también tienden a comprar Y.

Algunos ejemplos populares de algoritmos de aprendizaje no supervisados son:

- k-means para problemas de agrupamiento.
- Algoritmo de Apriori para problemas de aprendizaje de reglas de asociación.

3.3 APRENDIZAJE SEMI SUPERVISADO

Los problemas en los que tiene una gran cantidad de datos de entrada (X) y solo algunos de los datos están etiquetados (Y) se denominan problemas de aprendizaje semi supervisados.

Estos problemas se encuentran entre el aprendizaje supervisado y el no supervisado.

Un buen ejemplo es un archivo de fotos donde solo algunas de las imágenes están etiquetadas (por ejemplo, perro, gato, persona) y la mayoría no están etiquetadas.

Muchos problemas del aprendizaje automático en el mundo real entran en esta área. Esto se debe a que puede ser muy costoso o llevar mucho tiempo etiquetar los datos, ya que puede requerir

acceso a expertos en dominios. Mientras que los datos sin etiquetar son baratos y fáciles de recopilar y almacenar.

Puede utilizar técnicas de aprendizaje sin supervisión para descubrir y aprender la estructura en las variables de entrada.

También puede usar técnicas de aprendizaje supervisado para hacer mejores predicciones para los datos no etiquetados, alimentar esos datos al algoritmo de aprendizaje supervisado como datos de entrenamiento y usar el modelo para hacer predicciones sobre nuevos datos no vistos.

3.4 APRENDIZAJE REFORZADO

Esta modalidad es menos común y mucho más compleja, pero genera resultados increíbles. No usa etiquetas como tales, y en cambio usa recompensas para aprender.

Es como el aprendizaje por refuerzo de la psicología. En este enfoque, se utilizan comentarios positivos y negativos ocasionales para reforzar los comportamientos. Es como entrenar a un perro, los buenos comportamientos son recompensados con un regalo y se vuelven más comunes. Los malos comportamientos son castigados y se vuelven menos comunes. Este comportamiento motivado por la recompensa es clave en el aprendizaje de refuerzo.

A veces se ve el aprendizaje de refuerzo como “el aprender de los errores”. Si se coloca un algoritmo de aprendizaje de refuerzo en cualquier entorno, al principio, se cometerán muchos errores. Si se proporciona algún tipo de señal al algoritmo que asocie buenos comportamientos con una señal positiva y malos comportamientos con uno negativo, podemos reforzar nuestro algoritmo para preferir los buenos comportamientos a los malos. Con el tiempo, nuestro algoritmo de aprendizaje aprende a cometer menos errores de lo que solía hacerlo.

4 ÁRBOL DE DECISIÓN

Los algoritmos de aprendizaje basados en árboles se consideran uno de los mejores y más utilizados métodos de aprendizaje supervisado. Los métodos basados en árboles potencian los modelos predictivos con alta precisión, estabilidad y facilidad de interpretación.

A diferencia de los modelos lineales, mapean bastante bien las relaciones no lineales. Son adaptables para resolver cualquier tipo de problema, ya sean de clasificación o de regresión.

El proceso de clasificar a los clientes en un grupo de clientes potenciales y no potenciales o solicitudes de préstamos seguros o riesgosos se conoce como un problema de clasificación. La clasificación es un proceso de dos pasos, paso de aprendizaje y paso de predicción. En el paso de aprendizaje, el modelo se desarrolla en base a datos de capacitación dados.

En el paso de predicción, el modelo se usa para predecir la respuesta para datos dados. El Árbol de decisión en Machine Learning es uno de los algoritmos de clasificación más fáciles y populares de entender e interpretar.

Un árbol de decisión en Machine Learning es una estructura de árbol similar a un diagrama de flujo donde un nodo interno representa una característica (o atributo), la rama representa una regla de decisión y cada nodo hoja representa el resultado.

El nodo superior en un árbol de decisión en Machine Learning se conoce como el nodo raíz. Aprende a particionar en función del valor del atributo. Divide el árbol de una manera recursiva llamada partición recursiva.

Esta estructura tipo diagrama de flujo lo ayuda a tomar decisiones. Es una visualización como un diagrama de flujo que imita fácilmente el pensamiento a nivel humano. Es por eso que los árboles de decisión son fáciles de entender e interpretar.

Los árboles de decisión clasifican los ejemplos clasificándolos por el árbol desde la raíz hasta algún nodo hoja, con el nodo hoja proporcionando la clasificación al ejemplo, este enfoque se llama Enfoque de arriba hacia abajo.

Cada nodo en el árbol actúa como un caso de prueba para algún atributo, y cada borde que desciende de ese nodo corresponde a una de las posibles respuestas al caso de prueba. Este proceso es recursivo y se repite para cada subárbol enraizado en los nuevos nodos.

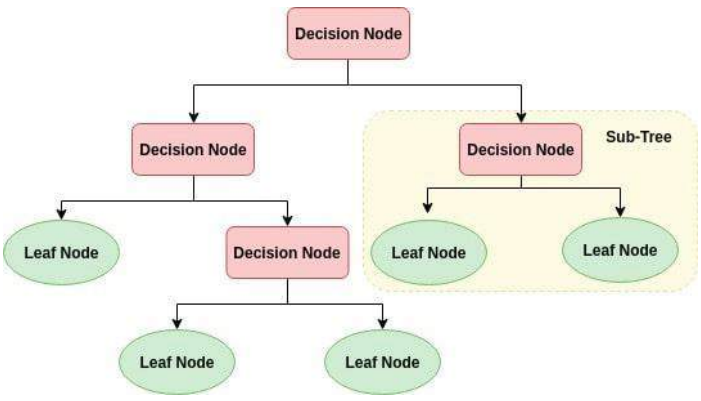


Ilustración 1 Árbol de decisión

Antes de profundizar más en los árboles de decisión vamos a familiarizarnos con algunas terminologías típicas:

- **Nodo raíz (nodo de decisión superior):** Representa a toda la población o muestra y este se divide en dos o más conjuntos homogéneos.
- **División:** Es un proceso de división de un nodo en dos o más subnodos.
- **Nodo de decisión:** Cuando un subnodo se divide en subnodos adicionales, se llama nodo de decisión.
- **Nodo de hoja / terminal:** Los nodos sin hijos (sin división adicional) se llaman Hoja o nodo terminal.
- **Poda:** Cuando reducimos el tamaño de los árboles de decisión eliminando nodos (opuesto a la división), el proceso se llama poda.
- **Rama / Subárbol:** Una subsección del árbol de decisión se denomina rama o subárbol.
- **Nodo padre e hijo:** Un nodo, que se divide en subnodos se denomina nodo principal de subnodos, mientras que los subnodos son hijos de un nodo principal.

4.1 FUNCIONAMIENTO DEL ALGORITMO

La idea básica detrás de cualquier algoritmo de árbol de decisión es la siguiente:

Seleccionar el mejor atributo utilizando Medidas de selección de atributos (ASM) para dividir los registros.

Hacer que ese atributo sea un nodo de decisión y divida el conjunto de datos en subconjuntos más pequeños recursivamente para cada hijo hasta que una de las condiciones coincida:

- Todas las tuplas pertenecen al mismo valor de atributo.
- No quedan más atributos.
- No hay más instancias.

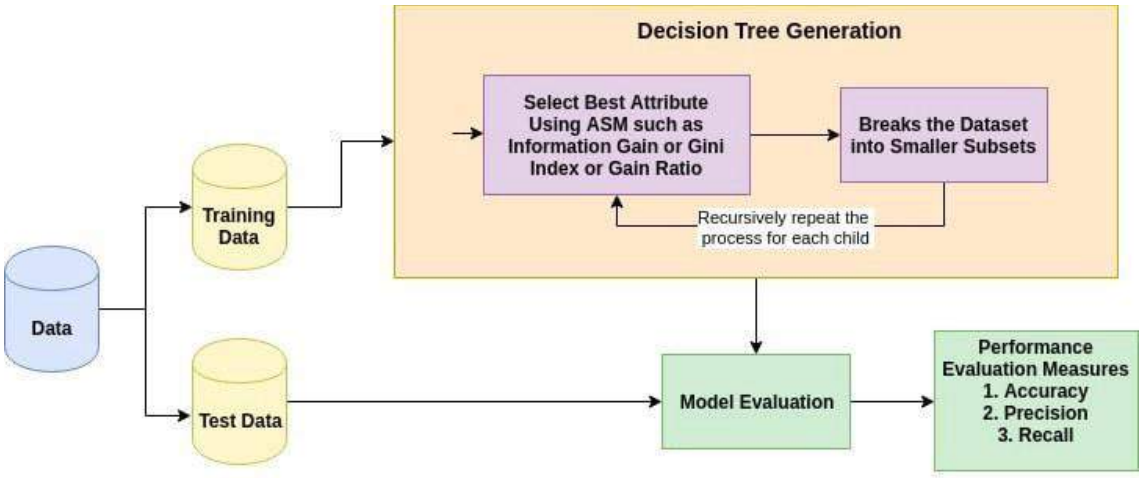


Ilustración 2 Funcionamiento árbol de decisión

4.2 VENTAJAS Y DESVENTAJAS

A continuación, se mostrarán las principales ventajas que tienen los árboles de decisión en el aprendizaje automático:

1. Los árboles de decisión se pueden usar para la regresión o la clasificación, aunque son mucho más populares para los problemas de clasificación. En general, si desea usar un árbol de decisión para un modelo de regresión, se recomienda usar un método de conjunto.
2. Los árboles de decisión no son paramétricos, lo cual es una manera elegante de decir que no se están haciendo suposiciones sobre cómo se distribuyen nuestros datos y la estructura (parámetros) de nuestro modelo se determinará a partir de la entrada del usuario y las observaciones en nuestra muestra, es decir, el modelo cogerá la forma gracias a los datos. Los modelos no paramétricos son excelentes cuando tenemos muchos datos.
3. Los árboles de decisión son interpretables, lo que significa que después de construir el modelo, podemos interpretar el modelo, no solo las predicciones.
4. Los árboles de decisión son rápidos porque son simples. Si bien es posible que no prioricemos la velocidad al construir un modelo final, esto puede ser una gran ventaja si solo estamos tratando de construir un modelo para obtener una comprensión inicial de nuestros datos.
5. Finalmente, el preprocesamiento de datos es más fácil con los árboles de decisión porque no tenemos que escalar nuestros datos. En algunas librerías de Python, nos vemos obligados a codificar las variables categóricas.

Como hemos visto, hay muchos aspectos positivos al usar árboles de decisión. Pero en determinadas situaciones puede que no sea la mejor opción. Si tenemos un tamaño de muestra pequeño, y para la regresión, puede que no sea la mejor opción si creemos que vamos a predecir valores finales que estarán fuera de los que contiene nuestra muestra de entrenamiento. Además de estos puntos, los árboles de decisión, como todos los modelos, también tienen algunas desventajas:

1. La desventaja más importante de los árboles de decisión es que son propensos al sobreajuste. Los árboles de decisión se sobre ajustan porque puede terminar con un nodo hoja para cada valor objetivo en sus datos de entrenamiento.
2. Los árboles de decisión también están localmente optimizados, lo que significa que no piensan en el futuro cuando deciden cómo dividirse en un nodo determinado. Solo buscan minimizar el valor escogido para la selección de atributos.
3. Debido a la naturaleza de la división, las clases desequilibradas también plantean un problema importante para los árboles de decisión cuando se trata de la clasificación. En cada división, el árbol decide cómo dividir mejor las clases en los siguientes dos nodos. Entonces, cuando una clase tiene una representación muy baja (la clase minoritaria), muchas de esas observaciones pueden perderse en los nodos de la clase mayoritaria, y luego la predicción de la clase minoritaria será aún menos probable de lo que debería, si alguno de los nodos la predice.

4.3 CONTROLAR EL TAMAÑO DEL ÁRBOL

El tamaño final que adquiere un árbol puede controlarse mediante reglas de parada que detengan la división de los nodos dependiendo de si se cumplen o no determinadas condiciones. El nombre de estas condiciones puede variar dependiendo del software o librería empleada, pero suelen estar presentes en todos ellos.

- **Observaciones mínimas para división**: define el número mínimo de observaciones que debe tener un nodo para poder ser dividido. Cuanto mayor el valor, menos flexible es el modelo.
- **Observaciones mínimas de nodo terminal:** define el número mínimo de observaciones que deben tener los nodos terminales. Su efecto es muy similar al de observaciones mínimas para división.
- **Profundidad máxima del árbol:** define la profundidad máxima del árbol, entendiendo por profundidad máxima el número de divisiones de la rama más larga (en sentido descendente) del árbol.
- **Número máximo de nodos terminales:** define el número máximo de nodos terminales que puede tener el árbol. Una vez alcanzado el límite, se detienen las divisiones. Su efecto es similar al de controlar la profundidad máxima del árbol.
- **Reducción mínima de error:** define la reducción mínima de error que tiene que conseguir una división para que se lleve a cabo.

Todos estos parámetros son lo que se conoce como hiperparámetros porque no se aprenden durante el entrenamiento del modelo. Su valor tiene que ser especificado por el usuario en base a su conocimiento del problema y mediante el uso de validación cruzada.

5 MEDIDAS DE SELECCIÓN DE ATRIBUTOS

La medida de selección de atributos es una heurística para seleccionar el criterio de división que divide los datos de la mejor manera posible. También se conoce como reglas de división porque nos ayuda a determinar puntos de interrupción para tuplas en un nodo dado.

ASM proporciona un rango para cada característica (o atributo) al explicar el conjunto de datos dado. El atributo de mejor puntuación se seleccionará como un atributo de división.

En el caso de un atributo de valor continuo, los puntos de división para las ramas también deben definirse. Las medidas de selección más populares son ganancia de información e índice de Gini.

5.1 GANANCIA DE INFORMACIÓN

La ganancia de información es una propiedad estadística que mide qué tan bien un atributo dado separa los ejemplos de entrenamiento de acuerdo con su clasificación objetivo.

Si se observa la ilustración 3 y se piensa qué nodo se puede describir fácilmente. La respuesta siempre seria C porque requiere menos información ya que todos los valores son similares.

Por otro lado, B requiere más información para describirlo y A requiere la información máxima. En otras palabras, podemos decir que C es un nodo puro, B es menos impuro y A es más impuro.

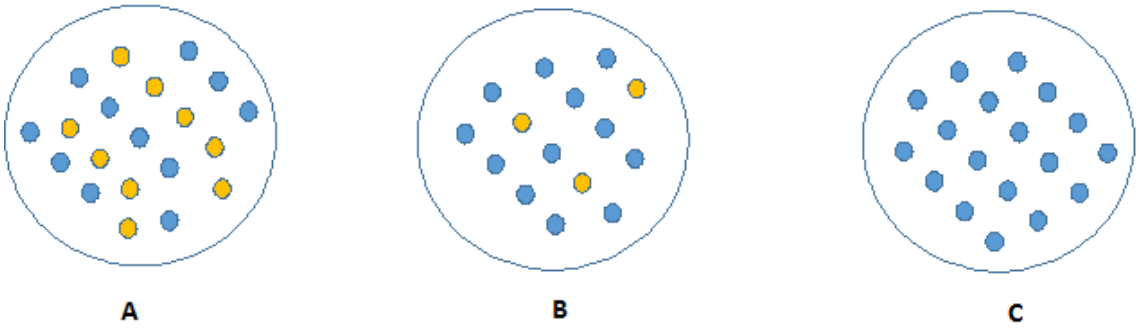


Ilustración 3 Ganancia de información

Podemos llegar a la conclusión de que un nodo menos impuro requiere menos información para describirlo. Y, el nodo más impuro requiere más información.

En un ejemplo paralelo considerando la ganancia de información, en la ilustración 4, a continuación, podemos ver que un atributo con baja ganancia de información (izquierda) divide los datos de manera relativamente uniforme y como resultado no obtenemos una decisión clara.

Mientras que un atributo con alta ganancia de información (derecha) divide los datos en grupos con un número desigual de positivos y negativos y, como resultado, ayuda a separarlos entre sí.

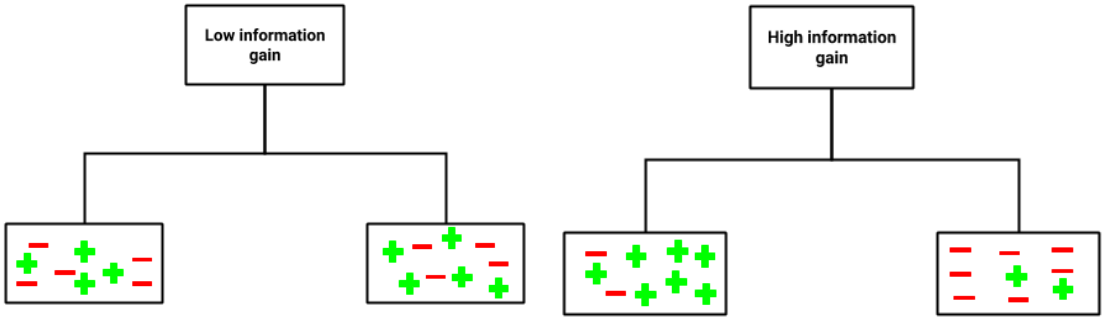


Ilustración 4 Ganancia de poca y alta información

Los pasos para calcular la ganancia de información son:

1. Calcular la entropía del nodo principal
2. Calcular la entropía de cada nodo individual de división y calcular el promedio ponderado de todos los subnodos disponibles en una división.
3. Calcular la ganancia de la información

Para poder calcular la ganancia de información, primero tenemos que introducir el término entropía de un conjunto de datos.

5.1.1 Entropía

Shannon inventó el concepto de entropía, que mide la impureza del conjunto de entrada. En física y matemáticas, la entropía se conoce como aleatoriedad o impureza en el sistema. En teoría de la información, se refiere a la impureza en un grupo de ejemplos. La ganancia de información es una disminución de la entropía.

La idea detrás de la entropía es, en términos simplificados, la siguiente: imaginar que se tiene una rueda de lotería que incluye 100 bolas verdes. Se puede decir que el conjunto de bolas dentro de la rueda de lotería es totalmente puro porque solo se incluyen bolas verdes.

Para expresar esto en la terminología de entropía, este conjunto de bolas tiene una entropía de 0 (también podemos decir impureza cero). Considere ahora, 30 de estas bolas son reemplazadas por bolas rojas y 20 por bolas azules como se puede apreciar en la ilustración 5:

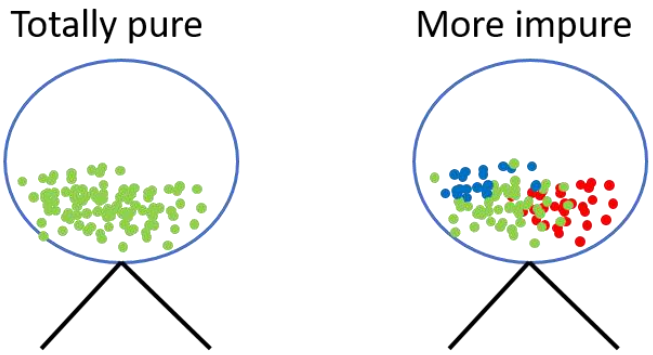


Ilustración 5 Puridad de datos

Si ahora se saca otra bola de la rueda de lotería, la probabilidad de recibir una bola verde se ha reducido de 1.0 a 0.5. Como la impureza aumentó, la pureza disminuyó, por lo tanto, también aumentó la entropía.

Por lo tanto, podemos decir que cuanto más "impuro" sea un conjunto de datos, mayor será la entropía y cuando menos "impuro" un conjunto de datos, menor será la entropía

Se debe tener en cuenta que la entropía es 0 si todos los miembros de S pertenecen a la misma clase. Por ejemplo, si todos los miembros son positivos, Entropía (S) = 0.

La entropía es 1 cuando la muestra contiene el mismo número de ejemplos positivos y negativos. Si la muestra contiene un número desigual de ejemplos positivos y negativos, la entropía está entre 0 y 1.

La siguiente ilustración 6 muestra la forma de la función de entropía en relación con una clasificación booleana, ya que la entropía varía entre 0 y 1.

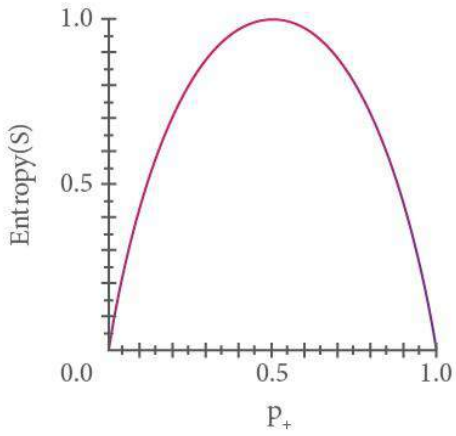


Ilustración 6 función de entropía en relación con una clasificación booleana

Se puede calcular la entropía usando la fórmula:

$$Entropia = \ -p * log_2 p - q * log_2 q$$

Donde p y q es la probabilidad de éxito y fracaso respectivamente en ese nodo. La entropía también se usa con la variable objetivo categórica. Se elige la división que tiene la entropía más baja en comparación con el nodo principal y otras divisiones. Cuanto menor sea la entropía, mejor será.

La ganancia de información calcula la diferencia entre la entropía antes de la división y la entropía promedio después de la división del conjunto de datos en función de los valores de atributo dados.

$$Ganancia\ de\ información = \ \ Entropía\ nodo\ padre - [Average\ Entropia(nodo\ hijo)]$$

Vamos a ver un ejemplo sobre como calcular la ganancia de la información:

Suponemos que se dispone de una muestra de 30 estudiantes con tres variables Sexo (Niño / Niña) y Clase (IX / X). 15 de estos 30 juegan al cricket en el tiempo libre.

Ahora, se quiere crear un modelo para predecir quién jugará al cricket durante un período de ocio. En este problema, necesitamos segregar a los estudiantes que juegan al cricket en su tiempo libre según una variable de entrada altamente significativa de entre las dos.

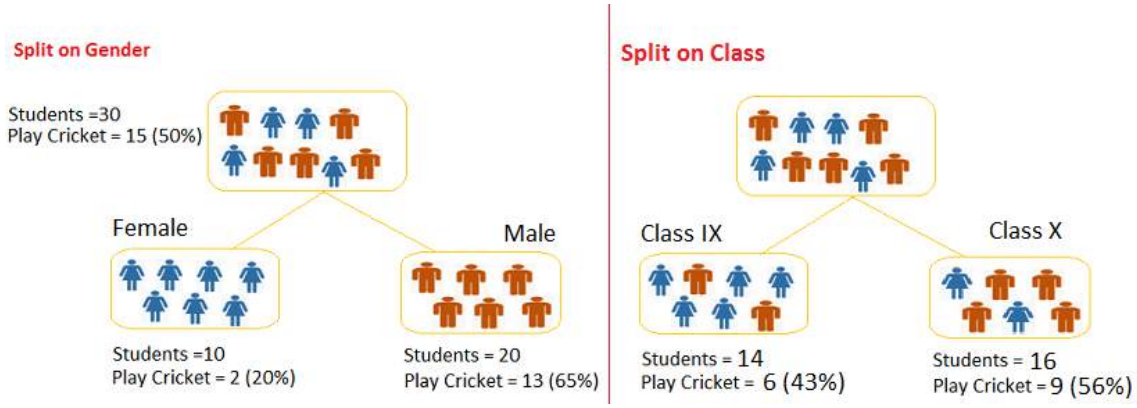


Ilustración 7 Ejemplo entropía

Entropía para el nodo principal = – (15/30) log2 (15/30) – (15 / 30) log2 (15/30) = 1

Claro ejemplo de que nos encontramos con una muestra que es un nodo impuro.

Para dividir en género:

- Entropía para nodo femenino = – (2/10) log2 (2/10) – (8/10) log2 (8/10) = 0,72
- Entropía para el nodo masculino = – (13/20) log2 (13/20) – (7/20) log2 (7/20) = 0.93
- Entropía por división Género = (10/30) * 0.72 + (20/30) * 0.93 = 0.86
- Ganancia de información por división por género = 1–0.86 = **0.14**

Para división en clase:

- Entropía para nodo de clase IX = – (6/14) log2 (6/14) – (8/14) log2 (8/14) = 0.99
- Entropía para clase Nodo X = – (9/16) log2 (9/16) – (7/16) log2 (7/16) = 0.99
- Entropía para división Clase = (14/30) * 0.99 + (16/30) * 0.99 = 0,99
- Ganancia de información por división por clase = 1– 0.99 = **0.01**

Arriba, puede ver que la ganancia de información para la separación por género es la más alta de todas, por lo que el árbol se dividirá en género.

5.2 GINI

Gini dice que, si seleccionamos dos elementos de una población al azar, entonces deben ser de la misma clase y la probabilidad de esto es 1 si la población es pura. Las características de la selección de atributo Gini son:

- Funciona con la variable objetivo categórica “Éxito” o “Fracaso” “.
- Realiza solo divisiones binarias
- Cuanto mayor sea el valor de Gini, mayor será la homogeneidad.
- CART (árbol de clasificación y regresión) utiliza el método Gini para crear divisiones binarias.

Los pasos para calcular Gini para una división son los siguientes:

1. Calcular Gini para subnodos, usando la suma de la fórmula del cuadrado de probabilidad de éxito y fracaso ($p^2 + q^2$). Donde p y q es la probabilidad de éxito y fracaso respectivamente en ese nodo.
2. Calcular Gini para la división usando la puntuación ponderada de Gini de cada nodo de esa división

A menudo se puede encontrar el término 'Impureza de Gini' que se determina restando el valor de Gini de 1. En términos generales, podemos decir:

$$Impureza\ de\ Gini = 1 - Gini$$

Gini es el parámetro predeterminado en el algoritmo del árbol de decisión, siendo asi, el método más utilizado en la selección de atributos.

Al igual que en caso de la ganancia de la información, vamos a mostrar un ejemplo de la selección de atributos con Gini:

Nos referiremos al ejemplo utilizado anteriormente, donde queremos segregar a los estudiantes en función de la variable objetivo (jugar cricket o no). En la siguiente figura, dividimos la población usando dos variables de entrada de género y clase. Los datos son exactamente iguales que en caso de la ganancia de la información. El siguiente paso es identificar qué división está produciendo subnodos más homogéneos usando Gini.

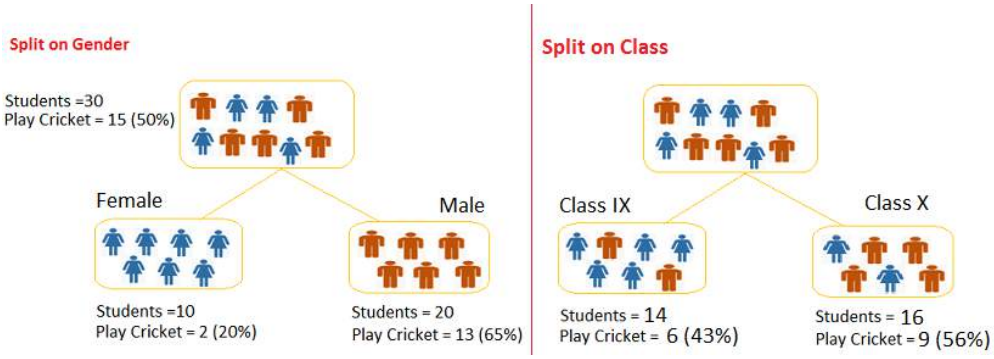


Ilustración 8 Ejemplo Gini

<u>División en género:</u>

- Calcular, Gini para el subnodo Hembra = (0.2) * (0.2) + (0.8) * (0.8) = 0.68
- Gini para sub-nodo Macho = (0.65) * (0.65) + (0.35) * (0.35) = 0.55
- Calcular Gini ponderado para Split Gender = (10/30) * 0.68+ (20/30) * 0.55 = **0.59**

<u>División en clase:</u>

- Gini para sub-nodo Clase IX = (0.43) * (0.43) + (0.57) * (0.57) = 0.51
- Gini para sub-nodo Clase X = (0.56) * (0.56) + (0.44) * (0.44) = 0.51
- Calcular Gini ponderado para clase dividida = (14 /30)*0.51+(16/30)*0.51 = **0.51**

Arriba, puede ver que la puntuación de Gini para dividir en género es mayor que dividir en clase, por lo tanto, la división de nodos tendrá lugar en género.

6 MÉTODOS DE CONJUNTO

Desde hace años, los árboles de decisión han dejado de ser unos algoritmos importantes, usados o precisos en el mundo del aprendizaje automático. En su lugar, han aparecido los métodos de conjunto.

6.1 DEFINICIÓN

Los métodos de conjunto son técnicas que crean múltiples modelos y luego los combinan para producir mejores resultados. Los métodos de conjunto generalmente producen soluciones más precisas que un solo modelo. Este ha sido el caso en una serie de concursos de aprendizaje automático, donde las soluciones ganadoras utilizaron métodos de conjunto. En la popular competencia de Netflix, el ganador utilizó un método de conjunto para implementar un poderoso algoritmo de filtrado colaborativo. Otro ejemplo es KDD 2009, donde el ganador también utilizó métodos de conjunto.

En concreto, para los métodos de conjunto de árboles de decisión se combinan varios árboles de decisión para producir un mejor rendimiento predictivo que utilizar un solo árbol de decisión. Además, se reduce uno de los principales problemas del árbol de decisión, que es el sobre ajuste. El principio principal detrás del modelo de conjunto es que muchos aprendizajes débiles se unen para formar así un aprendizaje fuerte. Sin embargo, estos métodos de conjunto pueden volverse bastante complejos y convertirse en modelos de caja negra, es decir, que perdamos la interpretación lógica del modelo.

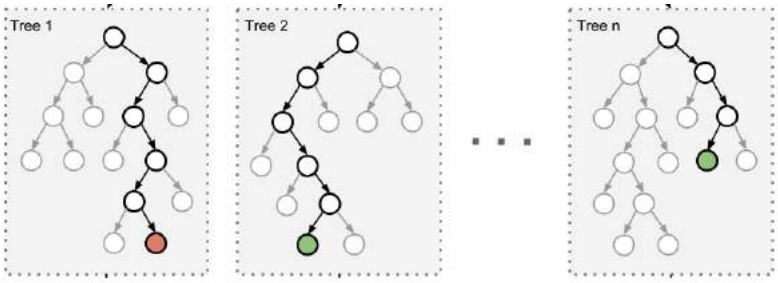


Ilustración 9 Métodos de conjuntos

Hablaremos de las siguientes técnicas de métodos de conjunto de los árboles de decisión:

- **Bagging (Bootstrap Aggregation):** se usa cuando nuestro objetivo es reducir la varianza de un árbol de decisión. Aquí la idea es crear varios subconjuntos de datos a partir de una muestra de entrenamiento elegida al azar con reemplazo. Cada colección de datos de subconjunto se usa para entrenar sus árboles de decisión. Como resultado, se obtiene un conjunto de diferentes modelos. Se utiliza el promedio de todas las predicciones de los diferentes árboles obtenidos, consiguiendo así una robustez en el modelo superior que un solo árbol de decisión.
- **Random Forest:** es una extensión del método bagging. Se añade un paso adicional, donde además de tomar el subconjunto aleatorio de datos, también toma la selección aleatoria de atributos en lugar de usarlos todos para crear los árboles.

- **Boosting:** es otra técnica de conjunto para crear un conjunto de predictores. La idea detrás de boosting es ajustar, de forma secuencial, múltiples weak learners (modelos sencillos que predicen solo ligeramente mejor que lo esperado por azar). Cada nuevo modelo emplea información del modelo anterior para aprender de sus errores, mejorando iteración a iteración. En otras palabras, ajustamos árboles consecutivos (muestra aleatoria) y en cada paso, el objetivo es resolver el error neto del árbol anterior.

6.2 BOOTSTRAPPING

Antes de entrar en profundidad con cada una de las técnicas de conjuntos, debemos definir el termino muestreo repetido o Bootstrapping en inglés. Este término es muy importante tanto en la técnica de bagging como en la del Random Forest.

Bootstrap se refiere al re muestreo aleatorio con reemplazo. Bootstrap nos permite comprender mejor el sesgo y la varianza con el conjunto de datos. Bootstrap implica un muestreo aleatorio de un pequeño subconjunto de datos del conjunto de datos inicial. Este subconjunto puede ser reemplazado. La selección de todos los ejemplos en el conjunto de datos tiene la misma probabilidad. Remplazo significa que, si un dato ha sido seleccionado para un subconjunto, este mismo, puede volver a ser elegido en el mismo subconjunto. Este método puede ayudar a comprender mejor la media y la desviación estándar del conjunto de datos.

Ilustración 10 Ejemplo de bootstrapping

La hipótesis que debe verificarse para que esta aproximación sea válida es doble:

- **Representatividad:** El tamaño N del conjunto de datos inicial debe ser lo suficientemente grande como para capturar la mayor parte de la complejidad de la distribución subyacente, de modo que el muestreo del conjunto de datos sea una buena aproximación del muestreo de la distribución real.
- **Independencia:** El tamaño N del conjunto de datos debe ser lo suficientemente grande en comparación con el tamaño B de las muestras de arranque para que las muestras no estén demasiado correlacionadas.

El muestreo de Bootstrap se usa en los algoritmos de conjunto de aprendizaje automático bagging y Random Forest. Con esta técnica se ayuda a evitar el sobreajuste y mejora la estabilidad de los algoritmos de aprendizaje automático.

7 DECISION TREE BAGGING

Al entrenar un modelo, no importa si estamos lidiando con una clasificación o un problema de regresión, obtenemos una función que toma una entrada, devuelve una salida y que ha aprendido de un conjunto de datos de entrenamiento. Debido a la variación teórica del conjunto de datos de entrenamiento, el modelo ajustado también está sujeto a variabilidad: si se hubiera observado otro conjunto de datos, habríamos obtenido un modelo.

7.1 DEFINICIÓN TEÓRICA

La idea del bagging es simple: queremos ajustar varios modelos independientes y "promediar" sus predicciones para obtener un modelo con una varianza más baja. Sin embargo, en la práctica, es prácticamente imposible adaptar modelos totalmente independientes porque requeriría demasiados datos. Por lo tanto, confiamos en las propiedades de las muestras de bootstrap (si se verifica la representatividad y la independencia) para ajustar modelos que son casi independientes.

Primero, creamos múltiples muestras de bootstrap para que cada nueva muestra de bootstrap actúe como otro conjunto de datos independiente extraído de la distribución verdadera. Luego, podemos crear un modelo débil de árbol de decisión para cada una de estas muestras y finalmente agregarlos de tal manera que "promediemos" sus resultados y, por lo tanto, obtengamos un modelo de conjunto con menos varianza que sus componentes.

Se puede decir que, como las muestras de bootstrap son aproximadamente independientes y distribuidas de manera idéntica, también lo son los modelos débiles aprendidos. Entonces, "promediar" los resultados de los modelos débiles no cambia la respuesta esperada, sino que reduce su varianza.

Al hacer bagging con árboles de decisión, nos podemos permitir tener árboles de decisión más sobre ajustados. Por esta razón, cada uno de los árboles se hacen profundos (es decir, pocas muestras de entrenamiento en cada nodo de hoja del árbol) y no se podan. Estos árboles tendrán una alta varianza y un bajo sesgo. Pero, así, mejoramos la eficacia del modelo bagging.

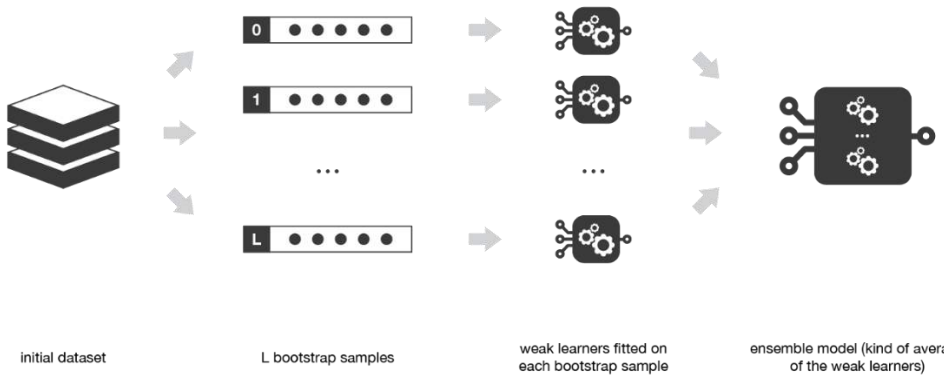


Ilustración 11 Funcionamiento bagging

La técnica de bagging se considera una técnica en paralelo, tal y como se muestra en la imagen de arriba en la ilustración 11, ya que cada modelo individual de árbol de decisión se crea únicamente con los datos seleccionados con boostrapping.

Finalmente, podemos mencionar que una de las grandes ventajas del bagging es que, como ya hemos dicho, trabaja en paralelo. Esto nos permite trabaja muy en detalle en cada uno de los modelos individuales e incluso, no permite utilizar diferentes tipos de algoritmos (combinar modelos de árboles de decisión con modelos de Naives Bayes por ejemplo).

7.2 VENTAJAS Y DESVENTAJAS

La técnica de bagging resulta muy eficiente en mucho caso. Las principales ventajas que aporta son:

- Reducir la varianza respecto un modelo único.
- Elimina el sobreajuste que puede producirse si se usa un único modelo.

Pero como cualquier técnica, también hay algunas desventajas:

- Introduce una pérdida de interpretabilidad.
- El modelo resultante puede experimentar muchos sesgos cuando se ignora el procedimiento adecuado.
- A pesar de que el ensacado es altamente preciso, puede ser computacionalmente costoso y esto puede desalentar su uso en ciertos casos.

8 RANDOM FOREST

El método Random Forest, tal y como su nombre lo indica, consiste en una gran cantidad de árboles de decisión individuales que operan como un conjunto, formando así un bosque. Dentro de este bosque, cada árbol individual genera una predicción de clase y la clase con más votos se convierte en la predicción de nuestro modelo. En la ilustración 12 se puede observar esta técnica:

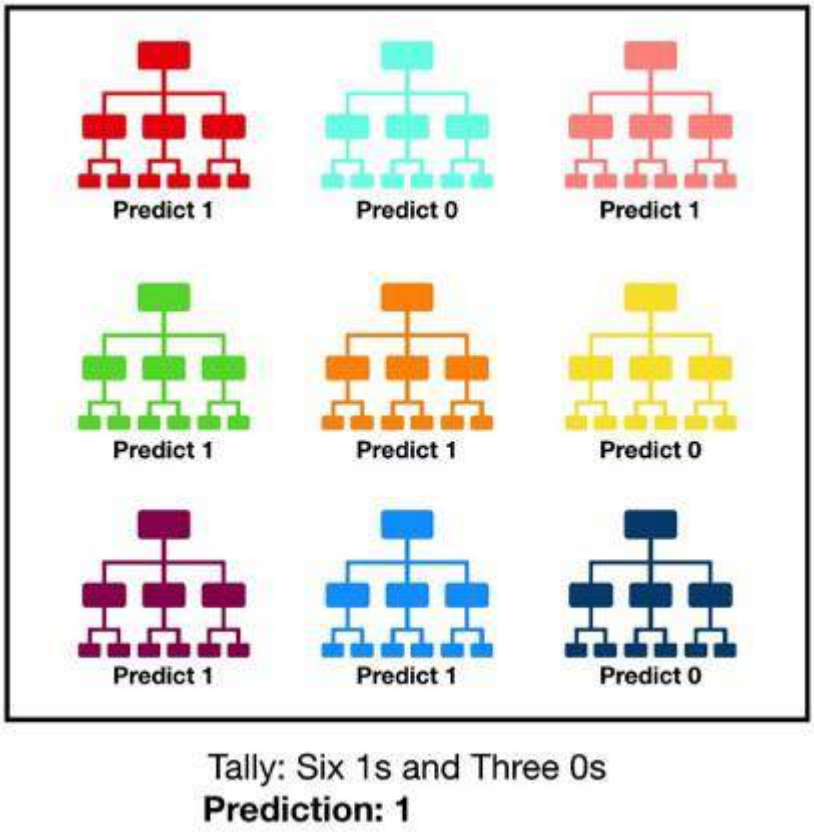


Ilustración 12 Random Forest

8.1 DEFINICIÓN TEÓRICA

El concepto fundamental detrás del Random Forest es simple pero poderoso: la sabiduría de las multitudes. En ciencia de datos, la razón por la que el modelo de bosque aleatorio funciona tan bien es que una gran cantidad de modelos (árboles) casi no correlacionados, que funcionan en grupo como comité, superarán en actuación a cualquiera de los modelos constituyentes individuales.

La baja correlación entre modelos es la clave. Los modelos no correlacionados pueden producir predicciones de conjunto que son más precisas que cualquiera de las predicciones individuales. La razón de este efecto es que los árboles se protegen entre sí de sus errores individuales (siempre que no se equivoquen constantemente en la misma dirección). Si bien algunos árboles pueden estar equivocados, muchos otros árboles estarán en lo correcto, por lo que, como grupo, los árboles se mueven en la dirección correcta.

Pero, ¿cómo se asegura el Random Forest de que el comportamiento de cada árbol individual no esté demasiado correlacionado con el comportamiento de ninguno de los otros árboles en el modelo? Para conseguirlo, utiliza los dos siguientes métodos:

- **Bagging:** los árboles de decisión son muy sensibles a los datos con los que han sido entrenados, pequeños cambios en el conjunto de entrenamiento pueden dar como resultado estructuras de árbol significativamente diferentes. El bosque aleatorio aprovecha esto al permitir que cada árbol individual muestree al azar del conjunto de datos con reemplazo, lo que da como resultado diferentes árboles.

  Hay que destacar que con el bagging no estamos dividiendo los datos iniciales en subconjuntos más pequeños. Más bien, si tenemos una muestra inicial de tamaño N, seguiremos alimentando a cada árbol débil con un conjunto de entrenamiento de tamaño N (a menos que se especifique lo contrario). Pero en lugar de los datos de entrenamiento originales, tomamos una muestra aleatoria de tamaño N con reemplazo. Por ejemplo, si nuestros datos de entrenamiento fueron [1, 2, 3, 4, 5, 6], entonces podríamos darle a uno de nuestros árboles la siguiente lista [1, 2, 2, 3, 6, 6]. Se observa que ambas listas son de longitud seis y que "2" y "6" se repiten en los datos de entrenamiento seleccionados al azar que damos a nuestro árbol, esta es la propiedad de remplazo.

- **Aleatoriedad de características o atributos (Feature Randomness):** en un árbol de decisión normal, cuando es el momento de dividir un nodo, consideramos todas las características posibles y elegimos la que produce la mayor separación entre las observaciones en el nodo izquierdo y las del nodo derecho (usando la técnica de selección de atributos que se haya definido). Por el contrario, cada árbol en un bosque aleatorio solo puede elegir las características de un subconjunto aleatorio. Esto provoca a una variación aún mayor entre los árboles débiles del modelo, factor que provoca como resultado una menor correlación entre los árboles débiles y una mayor diversificación.

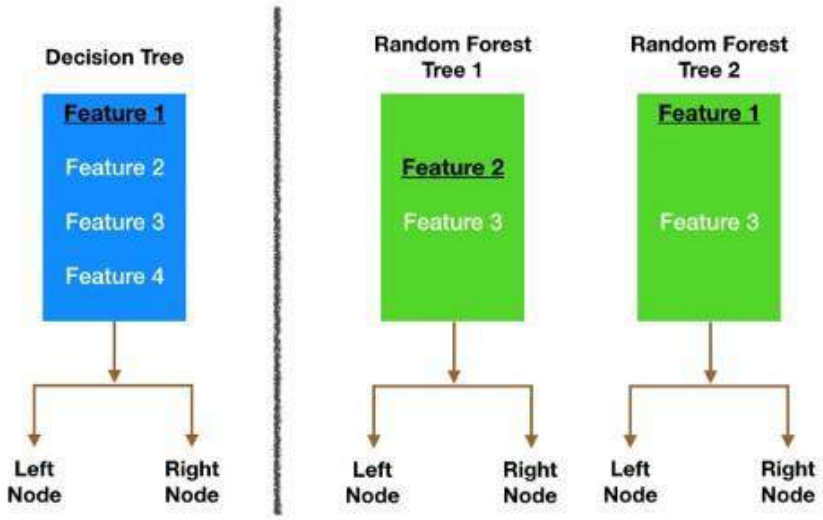


Ilustración 13 Aleatoriedad de atributos

Veamos un ejemplo visual: en la imagen de arriba, el árbol de decisión tradicional (en azul) puede seleccionar entre las cuatro características al decidir cómo dividir el nodo. Decide utilizar la función 1 (negra y subrayada), ya que divide los datos en grupos que están lo más separados posible.

Ahora se observa el Random Forest. Simplemente se observan dos de los árboles débiles del bosque en este ejemplo. Cuando se revisa el árbol débil aleatorio 1, se observa que este, solo puede considerar las características 2 y 3 (seleccionadas al azar) para su decisión de división de nodos. Sabemos por nuestro árbol de decisión tradicional (en azul) que la característica 1 es la mejor característica para dividir, pero el árbol 1 no puede ver la característica 1, por lo que se ve obligado a ir con la característica 2 (negra y subrayada). El árbol 2, por otro lado, solo puede ver las características 1 y 3, por lo que puede elegir la característica 1.

Entonces, en nuestro Random Forest, terminamos con árboles débiles que no solo están entrenados en diferentes conjuntos de datos (gracias al bagging) sino que también usan diferentes características para tomar sus decisiones. Este es el motivo por el cual los árboles débiles están muy poco correlacionados entre sí, pudiendo así, protegerse de sus errores.

8.2 VENTAJAS Y DESVENTAJAS

Random Forest se basa en el algoritmo de bagging y utiliza la técnica aprendizaje por conjunto. Crea tantos árboles en el subconjunto de datos y combina la salida de todos los árboles. De esta manera, se observará que algunas ventajas y desventajas coinciden con las del bagging. Las principales ventajas son:

- Reduce el problema de sobreajuste en los árboles de decisión, también reduce la variación y, como consecuencia, mejora la precisión
- Puede manejar hasta miles de variables de entrada e identificar las más significativas. Método de reducción de dimensionalidad.
- Existen muy pocas suposiciones y por lo tanto la preparación de los datos es mínima (no es necesario escalar los valores, por ejemplo)
- Incorpora métodos efectivos para estimar valores faltantes.
- Random Forest suele ser robusto para los valores atípicos y puede manejarlos automáticamente

Dentro de las desventajas del Random Forest se encontrarían:

- Pérdida de interpretación
- Poco control en lo que hace el modelo (modelo caja negra para modeladores estadísticos)
- Mayor complexidad del modelo, se necesitan muchos más arboles débiles. Este algoritmo requiere mucha más potencia y recursos computacionales. Por otro lado, el árbol de decisión es simple y no requiere tantos recursos computacionales.
- Se aumenta el período de entrenamiento, el Random Forest requiere mucho más tiempo para entrenar en comparación con los árboles de decisión normales, ya que genera muchos árboles y toma la decisión sobre la mayoría de los votos.

9 DECISIÓN TREE BOOSTING

Boosting significa combinar un algoritmo de aprendizaje en serie para lograr un modelo fuerte de muchos modelos débiles conectados secuencialmente. En nuestro caso, los modelos débiles serán arboles de decisión.

9.1 DEFINICIÓN TEÓRICA

La idea de este algoritmo es que cada árbol intenta minimizar los errores del árbol anterior. Cada árbol en el boosting es un modelo bastante débil, pero agregar muchos árboles débiles en serie y cada uno centrado en los errores del anterior hace que el boosting sea un modelo altamente eficiente y preciso. A diferencia del bagging, el boosting no implica el muestreo con bolsas. Cada vez que se agrega un nuevo árbol, este usa una versión levemente modificada del conjunto de datos iniciales.

Por lo tanto, la idea principal del boosting, al igual que el bagging, es combinar varios modelos débiles para generar un modelo más fuerte. La diferencia de los algoritmos Boosting es que crear los modelos débiles de manera secuencial, donde cada modelo intenta corregir los errores de su modelo predecesor tal y como se muestra en la ilustración 14.

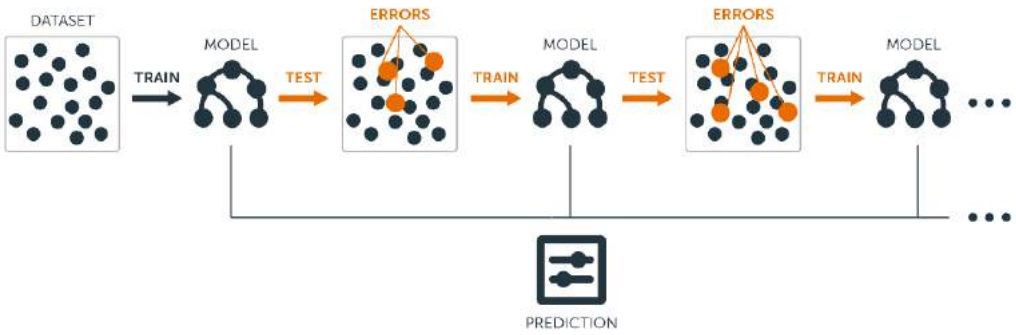


Ilustración 14 Boosting

Hay varios tipos de algoritmos que utilizan las técnicas de boosting. Todos ellos emplean la misma idea de realizar modelos débiles en serie, pero cada uno de ellos utilizan pequeñas técnicas diferentes. Los modelos más conocidos de boosting son:

- **AdaBoost (Adaptive Boosting):** el método que utiliza este algoritmo para corregir a su predecesor es prestando más atención a las instancias erróneas de entrenamiento del modelo anterior. Por lo tanto, en cada nuevo predictor, el modelo se centrará en los casos más que han sido erróneos en el modelo previo.
- **Gradient Boosting:** funciona muy igual que el AdaBoost. La diferencia radica en como ajusta los valores erróneos de su predecesor. Al contrario de AdaBoost, que modifica los pesos de las instancias en cada interacción, este método intenta ajustar el nuevo predictor a los errores residuales cometidos por el predictor anterior.
- **XG Boost algorithm:** es una implementación avanzada de Gradient Boosting. Este algoritmo tiene un alto poder predictivo y es diez veces más rápido que cualquier otra técnica de aumento de gradiente. Además, incluye una variedad de regularización que reduce el sobreajuste y mejora el rendimiento general.

9.2 ADABOOST

Ada-boost o Adaptive Boosting es un método de boosting propuestos por Yoav Freund y Robert Schapire en 1996. Como se ha visto, este método combina múltiples clasificadores débiles para aumentar la precisión del clasificador final. AdaBoost es un método de conjunto iterativo en serie.

El clasificador AdaBoost crea un clasificador fuerte combinando múltiples clasificadores de bajo rendimiento para que obtenga un clasificador fuerte de alta precisión. El concepto básico detrás de AdaBoost es establecer los pesos de los clasificadores y entrenar la muestra de datos en cada iteración de modo que garantice las predicciones precisas de observaciones inusuales anteriores. Cualquier algoritmo de aprendizaje automático se puede utilizar como clasificador base si acepta pesos en el conjunto de entrenamiento, con los árboles de decisión también se puede utilizar. AdaBoost debe cumplir dos condiciones:

1. El clasificador debe ser entrenado de manera interactiva en varios ejemplos de entrenamiento pesado.
2. En cada iteración, intenta proporcionar un ajuste excelente para estos datos minimizando el error de entrenamiento.

En la ilustración 15 se puede observar de manera gráfica como el método AdaBoost funciona de manera secuencial e intentando mejorar los errores del modelo anterior para, finalmente, conseguir un modelo muy robusto.

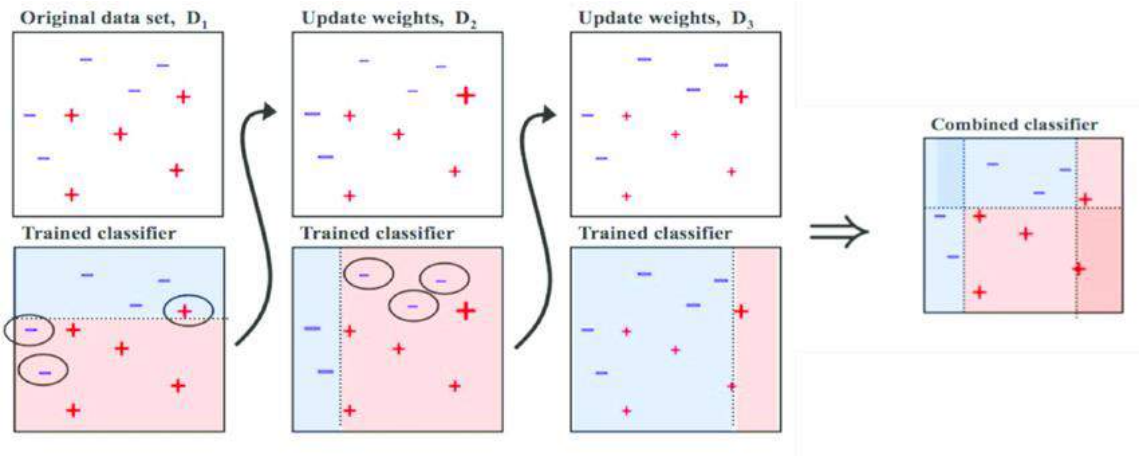


Ilustración 15 AdaBoost

A continuación, vamos a observar cómo funciona el algoritmo AdaBoost y cuáles son los pasos que sigue:

1. Inicialmente, AdaBoost selecciona un subconjunto de entrenamiento al azar.
2. Iterativamente, el modelo de aprendizaje automático AdaBoost va seleccionando el conjunto de entrenamiento basado en la predicción precisa del último entrenamiento.
3. Asigna el mayor peso a las observaciones clasificadas incorrectas para que en la próxima iteración estas observaciones obtengan la alta probabilidad de clasificación.
4. Además, asigna el peso al clasificador débil entrenado en cada iteración de acuerdo con la precisión del clasificador. El clasificador más preciso tendrá un alto peso.

5. Este proceso itera hasta que los datos de entrenamiento completos se ajusten sin ningún error o hasta que se alcance el número máximo de estimadores especificado.
6. Para clasificar, cada modelo débil creado realiza un "voto".

Las tres ideas principales que están detrás del algoritmo AdaBoost en árboles de decisión son:

1. Combina muchos modelos débiles para conseguir una clasificación final acurada. Cada árbol de decisión, forma un modelo débil, y este tiene una profundidad de 1. Es decir, son muy simples. Esta característica se puede llegar a ajustar si se precisa.

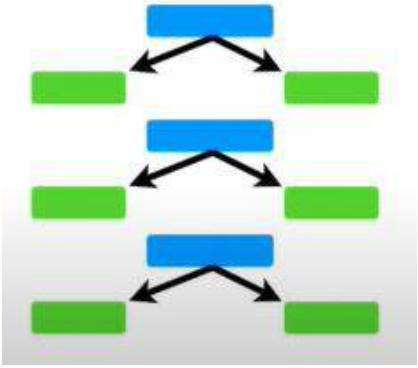


Ilustración 16 Arboles simples

2. Algunos de estos árboles de decisión tan débiles son más importantes que otros modelos débiles. Por esa razón, se ponderan los modelos débiles. De esta manera, en la votación final, algunos modelos débiles serán más importantes. La fórmula para ponderar un modelo es:

$$Ponderación = \frac{1}{2} \cdot \log(\frac{1 - Error\ total}{Error\ Total})$$

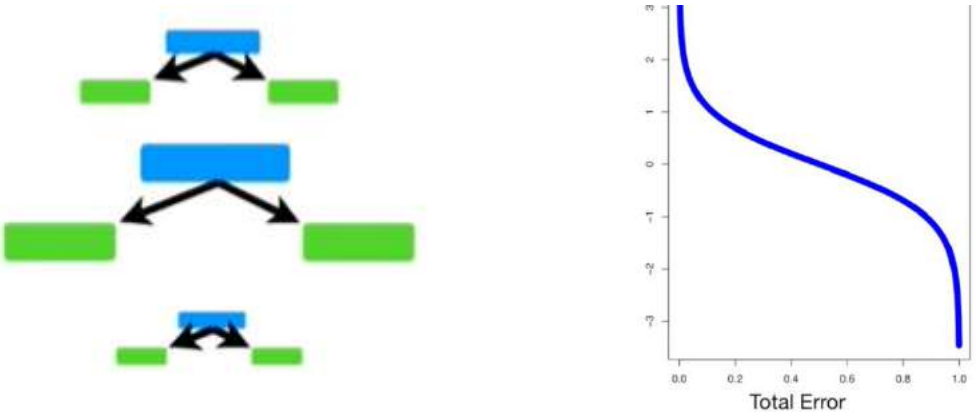


Ilustración 17 Ponderación de los modelos y su función

Donde el error total es la suma de los pesos de los datos que han sido clasificados incorrectamente. Vemos que, si el error es muy pequeño, el modelo tendrá una ponderación muy alta, por lo que se le dará una gran importancia a la hora de realizar las votaciones finales.

3. Cada uno de los árboles débiles, se lleva a cabo teniendo en cuenta los errores del modelo anterior. En cada iteración se asignan unos pesos a cada dato en función de si has sido predicho correctamente o no. Estos pesos finalmente se escalan par que entre todos sumen 1. La fórmula depende de si el dato ha sido incorrecto o correcto:

$$Nuevo\ Peso = peso \cdot e^{ponderación}$$
$$Nuevo\ Peso = peso \cdot e^{-ponderación}$$

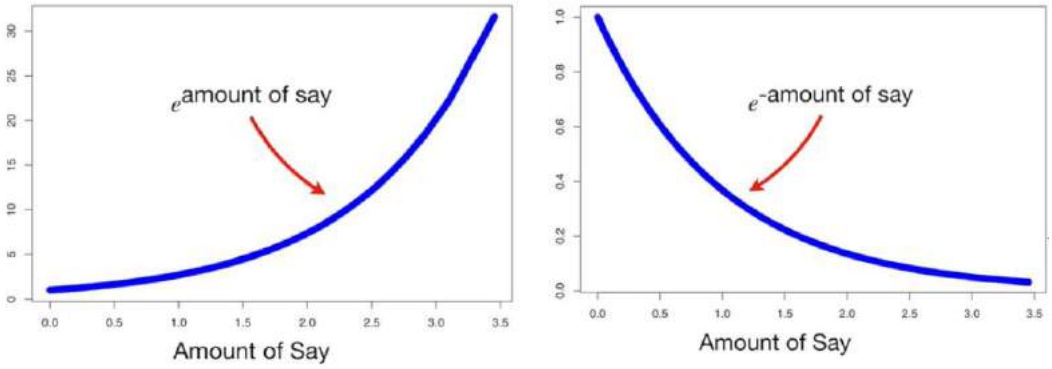


Ilustración 18 Reasignación de pesos de los datos

Se observa que cuando el nuevo dato ha sido incorrecto, el exponencial es positivo, consiguiendo así un valor mayor del nuevo peso. Cuando el dato es correcto, el nuevo peso decrece respecto al anterior.

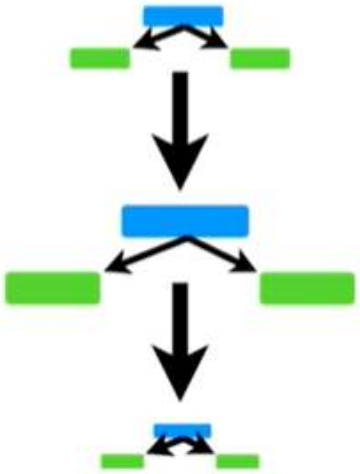


Ilustración 19 Evolución de los modelos en función de los modelos previos

9.3 VENTAJAS Y DESVENTAJAS

Las principales ventajas del algoritmo de boosting son:

- La simplicidad a la hora de implementarlo
- Consigue una muy buena generalización, perfecto para problemas de clasificación
- No es nada propenso al sobreajuste.

Pero como en cualquier modelo, no todo son ventajas. A continuación, mostramos las desventajas del modelo:

- Es muy sensible a lo datos erróneos y al ruido.
- Método difícil de racionalizar, ya que en cada iteración se hace una corrección.

10 LIBRERÍA SCIKIT-LEARN

Existen multitud de lenguajes informáticos: Python, Java, R, etc. Cualquiera de ellos proporciona una correcta base que será necesaria para trabajar con datos. Sin embargo, Python se define como uno de los lenguajes más intuitivos y eficientes.

Python es un lenguaje de programación interpretado, orientado a objetos de alto nivel y con semántica dinámica. Su sintaxis hace énfasis en la legibilidad del código, lo que facilita su depuración y, por tanto, favorece la productividad. Ofrece la potencia y la flexibilidad de los lenguajes compilados con una curva de aprendizaje suave.

Aunque Python fue creado como lenguaje de programación de uso general, cuenta con una serie de librerías y entornos de desarrollo para cada una de las fases del proceso del trabajo con datos. Esto, sumado a su potencia, su carácter open source y su facilidad de aprendizaje le ha llevado a tomar la delantera a otros lenguajes propios de la analítica de datos por medio del data science como pueden ser SAS (software comercial líder hasta el momento) y R (también open source, pero más propio de entornos académicos o de investigación).

Scikit-Learn es una de estas librerías gratuitas para Python. Cuenta con algoritmos de clasificación, regresión, clustering y reducción de dimensionalidad. Además, presenta la compatibilidad con otras librerías de Python como NumPy, SciPy y matplotlib.

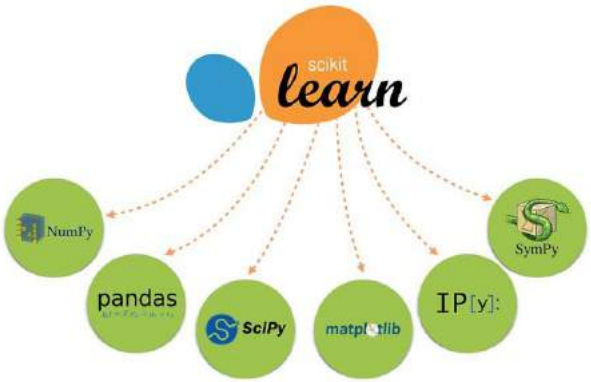


Ilustración 20 Librería Scikit-Learn

La gran variedad de algoritmos y utilidades de Scikit-learn la convierten en la herramienta básica para empezar a programar y estructurar los sistemas de análisis datos y modelado estadístico. Los algoritmos de Scikit-Learn se combinan y depuran con otras estructuras de datos y aplicaciones externas como Pandas o PyBrain.

La ventaja de la programación en Python, y Scikit-Learn en concreto, es la variedad de módulos y algoritmos que facilitan el aprendizaje y trabajo del científico de datos en las primeras fases de su desarrollo.

10.1 DECISION TREE

Para crear un árbol de decisión se puede usar la librería Scikit-Learn, en concreto, podemos importar el módulo "sklearn.tree.DecisionTreeClassifier". Una vez importado este módulo se deben definir los hiperparámetros que queremos que tenga. Los hiperparámetros son aquellos parámetros que son externos a los datos de entrenamiento, es decir, que son independientes. El módulo del árbol de clasificación tiene hasta 14 parámetros diferentes a definir. Entre todos estos parámetros, los más importantes y los que más destacan son:

- **criterion:** {"gini", "entropy"}, por defecto = "gini". La función para medir la calidad de una división. Los criterios admitidos son "gini" para la impureza de Gini y "entropía" para la ganancia de información.
- **splitter:** {"best", "random"}, por defecto = "best". La estrategia utilizada para elegir la división en cada nodo. Las estrategias admitidas son "best" para elegir la mejor división y "random" para elegir la mejor división aleatoria.
- **max_depth:** default = Ninguno. La profundidad máxima del árbol. Si decidimos "ninguno", los nodos se expanden hasta que todas las hojas sean puras o hasta que todas las hojas contengan menos de min_samples_split datos.
- **min_samples_split:** por defecto = 2. El número mínimo de muestras requeridas para dividir un nodo interno.
- **min_samples_leaf:** por defecto = 1. El número mínimo de muestras necesarias para que un nodo sea nodo hoja.
- **max_features:** por defecto = Ninguno. La cantidad máxima de características a considerar cuando se busca la mejor división.

Para poder escoger los valores correctos de los hiperparámetros se deben probar múltiples valores para ver cómo se comporta el modelo con cada uno. Se escogerán los valores que consigan un ajuste mejor del modelo.

10.2 DECISION TREE BAGGING

Al igual que con el decision tree, para el bagging también se puede importar un módulo de librería Scikit-Learn: "sklearn.ensemble.BaggingClassifier".

Para cada modelo de bagging que queramos entrenar deberemos definir previamente sus hiperparámetros. En total hay 11 diferentes, pero los más importantes y los que utilizaremos son:

- **base_estimator:** por defecto= Ninguno. Aquí es donde definiremos que tipos de modelo queremos que se entrenen por cada conjunto de datos. Si no se define ninguno, se tomará el algoritmo árbol de decisión.
- **n_estimators:** por defecto= 10. El número de modelos débiles por el cual estará formado el modelo final.
- **max_samples:** por defecto=1.Se indica el número de datos que compondrán cada subconjunto de datos. Se indica con un valor entre 0 y 1 que indica la proporción de datos en comparación al conjunto de datos inicial.
- **max_features:** por defecto=1. Indica el número de atributos que se tengan en cuenta a la hora de crear cada árbol débil. También es una proporción en función del número de

atributos que hay en los datos iniciales. El valor por defecto nos indica que tendremos en cuenta cada uno de los atributos iniciales.

Habrá algunos parámetros que no se tocarán en nuestro caso, como por ejemplo el base_estimator (ya que por defecto ya trabaja con árboles de decisión) y el max_features (ya que, si tocásemos este parámetro, nos estaríamos acercando mucho a la manera de funcionar del algoritmo Random Forest y alejando del bagging)

10.3 Random Forest

A continuación, vamos a ver cómo se puede importar el algoritmo Random Forest. También se usa la librería Scikit-Learn, en concreto el módulo: "sklearn.ensemble.RandomForestClassifier". Este es uno de los algoritmos que más hiperparámetros tiene, 19 en total. Al igual que en los casos anteriores, los que más se usan y más importantes son:

- **n_estimators:** por defecto= 100. El número de modelos débiles (arboles de decisión) por el cual estará formado el modelo final.
- **Criterion:** por defecto= "gini". Indica que función se utilizará para seleccionar el mejor atributo.
- **max_depth:** por defecto=ninguno. Indica la profundidad máxima que puede tener cada árbol debil
- **min_samples_split:** por defecto=2. Indica el valor mínimo de muestras que tiene que haber en un modo para poder realizar una separación de los datos.

Hay más hiperparámetros para ajustar el modelo un poco más. En nuestro caso no los usaremos ya que queremos realizar un estudio centrado en los modelos originales y no tan modificados.

10.4 Decision Tree Boosting (AdaBoost)

El último modelo que aplicaremos en Python es el decision tree Boosting, en concreto el AdaBoost. Como en los casos anteriores, el modelo se importa con "sklearn.ensemble. AdaBoostClassifier". Vemos que bagging, Random Forest y AdaBoost se importan del mismo submódulo "ensemble". AdaBoost tienen pocos hiperparámetros, tantos como 5. En este caso vamos a mencionar los todos:

- **base_estimator:** por defecto= ninguno. En este hiperparámetros se define que tipo de algoritmo se utilizara en los modelos débiles. Si no se define ninguno, se usa por defecto un árbol de decisión de profundidad máxima 1, es decir, DecisionTreeClassifier(max_depth=1).
- **n_estimators:** por defecto=50. El número que queremos que se genere de modelos débiles.
- **learning_rate:** por defecto=1. Nos marca la ratio de contribución que tendrá cada modelo débil al modelo final.
- **Algorithm:** por defecto= "SAMMER.R". Se marca como se implementará el algoritmo AdaBoost. Hay dos tipos únicamente. El que se utilizara es el de por defecto. Es importante

asegurare que el modelo que se haya escogido en base_estimator pueda funcionar con SAMMER.R algoritmo. En el caso del árbol de decisión sí que puede.

- **random_state:** por defecto= ninguno. Controlo la semilla aleatoria en cada una de los modelos débiles.

Los únicos hiperparámetros que se analizarán para el boosting serán base_estimator, n_estimators y el learning_rate. A los otros dos, se les asignara el valor por defecto.

11 DATA SET

En este apartado se comentará qué conjuntos de datos se han utilizado para realizar un estudio practico de cómo se comportan cada uno de los modelos siguientes: Decision Tree, Decision Tree Bagging, Random Forest y AdaBoost.

11.1 DATA SET ESCOGIDO

Para trabajar los algoritmos se ha escogido un Data set llamado: "Hotel booking demand". Este conjunto de datos está formado por más de 32 atributos y más de 100.000 instancias. Dichos datos han sido extraídos en formato *.csv de la web "Kaggle", un portal famoso por sus numerosos conjuntos de datos abiertos.

Este conjunto de datos contiene información de reservas para un hotel de ciudad y un hotel turístico e incluye información como cuándo se realizó la reserva, la duración de la estancia, si la reserva se canceló, la cantidad de adultos, niños y / o bebés, y la cantidad de espacios de estacionamiento de coches requeridos, entre otras cosas. Es importante destacar que toda la información de identificación personal ha sido eliminada de los datos.

Se sabe que ambos hoteles están ubicados en Portugal. El turístico se encuentra en la región turística de Algarve y el hotel de ciudad se encuentra en Lisboa. Los datos contienen reservas que deben llegar entre el 1 de julio de 2015 y el 31 de agosto de 2017.

A continuación, se muestra en más detalle cada uno de los atributos que forman nuestro data set:

- **ADR:** media del precio pagado por noche.
- **Adults:** número de adultos.
- **Agent:** identificador que muestra que agencia de viajes ha realizado la reserva. Es una variable categórica.
- **ArrivalDateDayOfMonth:** día del mes de la fecha de llegada.
- **ArrivalDateMonth:** mes de la fecha de llegada. Es una variable categórica con los nombres de meses.
- **ArrivalDateWeekNumber**: numero de la semana de la fecha de llegada.
- **ArrivalDateYear**: año de la fecha de llegada.
- **AssignedRoomType**: código del tipo de habitación que se asigna a la reserva.
- **Babies**: número de bebes.
- **BookingChanges**: número de cambios hechos en la reserva desde que reservo hasta que se hizo el check-in o la cancelación.
- **Children**: número de niños.
- **Company**: identificador de la compañía que hizo la reserva o que hizo el pagamento. Es una variable categorica.
- **Country**: indica el país de origen de los clientes. Es una variable categórica.
- **CustomerType**: el tipo de cliente. Puede ser de 4 tipos diferentes: contract, group, transient, transient-party. Es una variable categórica.

- **DaysInWaitingList**: Número de días que la reserva estuvo en lista de espera antes de ser confirmada al cliente.
- **DepositType**: indicador de si el cliente dejo un depósito para garantizar la reserva. Es una variable categórica.
- **DistributionChannel**: indicador de cuál fue el canal de distribución. Es una variable categórica.
- **Hotel**: indica a que hotel pertenece la reserva. Es una variable categórica.
- **IsCanceled**: valor que nos indica si una reserva fue cancelada (1) o no (0).
- **IsRepeatedGuest**: valor que indica si el nombre de la reserva forma parte de un cliente repetido (1) o no (0).
- **LeadTime**: número de días que hay entre la fecha de la reserva y la fecha prevista de entrada.
- **MarketSegment**: marca la designación del mercado. Es una variable categórica.
- **Meal**: tipo de comida que reservo. Es una variable categórica.
- **PreviousBookingNotCanceld**: número de reservas previas que no fueron canceladas por el cliente que está reservando.
- **PreviousCancellations**: número de reservas previas que fueron canceladas por el cliente que está reservando.
- **RequiredCarParkingSpacees**: número de plazas de parking requeridas por el cliente.
- **ReservationStatus**: se muestra el status de la reserva (puede ser: cancelled, check-out, no-show). Es una variable categórica.
- **ReservationStatusDate**: fecha en la que la última modificación de la reservationStatus fue hecha.
- **ReservedRoomType**: código del tipo de habitación que fue reservada. Es una variable categórica.
- **StaysInWeekengNights**: número de noches en fin de semana (sábado y domingo) que se han reservado
- **StaysInWeekNights**: número de noches reservadas entre semana.
- **TotalOfSpecialRequests**: número de requisitos pedido por el cliente.

Como se observa, hay tantos atributos numéricos como categóricos. Por lo que se deberán trabajar los datos previamente para poder usar algunos algoritmos.

Para entender un poco nuestro conjunto de datos, se adjunta la ilustración 21 con las cinco primeras instancias y sus respectivos atributos:

hotel	Resort Hotel	Resort Hotel	Resort Hotel	Resort Hotel	Resort Hotel
is_canceled	0	0	0	0	0
lead_time	342	737	7	13	14
arrival_date_year	2015	2015	2015	2015	2015
arrival_date_month	July	July	July	July	July
arrival_date_week_number	27	27	27	27	27
arrival_date_day_of_month	1	1	1	1	1
stays_in_weekend_nights	0	0	0	0	0
stays_in_week_nights	0	0	1	1	2
adults	2	2	1	1	2
children	0	0	0	0	0
babies	0	0	0	0	0
meal	BB	BB	BB	BB	BB
country	PRT	PRT	GBR	GBR	GBR
market_segment	Direct	Direct	Direct	Corporate	Online TA
distribution_channel	Direct	Direct	Direct	Corporate	TA/TO
is_repeated_guest	0	0	0	0	0
previous_cancellations	0	0	0	0	0
previous_bookings_not_canceled	0	0	0	0	0
reserved_room_type	C	C	A	A	A
assigned_room_type	C	C	C	A	A
booking_changes	3	4	0	0	0
deposit_type	No Deposit	No Deposit	No Deposit	No Deposit	No Deposit
agent	NaN	NaN	NaN	304	240
company	NaN	NaN	NaN	NaN	NaN
days_in_waiting_list	0	0	0	0	0
customer_type	Transient	Transient	Transient	Transient	Transient
adr	0	0	75	75	98
required_car_parking_spaces	0	0	0	0	0
total_of_special_requests	0	0	0	0	1
reservation_status	Check-Out	Check-Out	Check-Out	Check-Out	Check-Out
reservation_status_date	2015-07-01	2015-07-01	2015-07-02	2015-07-02	2015-07-03

Ilustración 21 Datos y atributos del problema

Una vez entendido cada uno de nuestros atributos correctamente y haber entendido el conjunto de datos, se ha definido cual va a ser nuestro objetivo para aplicar las técnicas de aprendizaje automático supervisado de clasificación.

El objetivo definido ha sido: predecir si una nueva reserva del hotel de ciudad será cancelada o no. Esta predicción se llevará acabo de manera instantánea cuando la reserva llegue por primera vez al hotel.

Hemos decidido centrarnos en el hotel de la ciudad para obtener un modelo más concreto que predice mejor las reservas de un hotel y no de dos. Se ha escogido el de ciudad ya que era el que tenía más instancias (79330 en el hotel ciudad y 40060 en el resort), y, además, tenía casi el mismo número de muestras canceladas que no canceladas tal y como se observa en la ilustración 22:

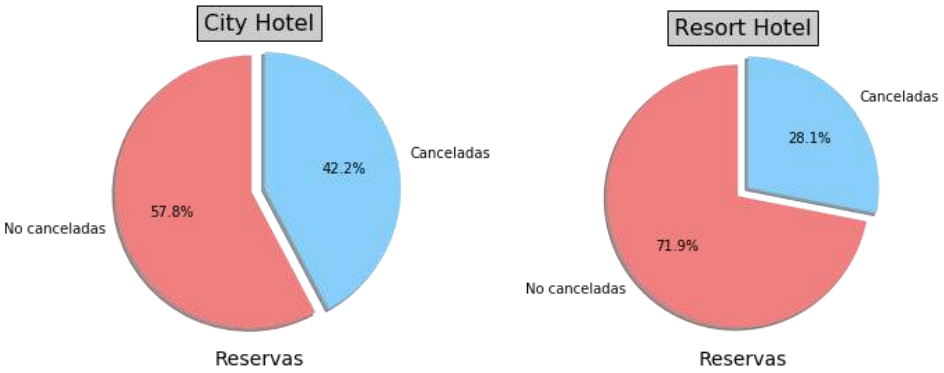


Ilustración 22 Proporción de reservas canceladas del "City Hotel" y del "Resort Hotel"

11.2 Preparación Data Set

En este apartado vamos estudiar el conjunto de datos de manera más precisa para eliminar aquellas instancias o atributos que sean incorrecto o que no tenga mucho sentido usarlos. De momento, nuestro conjunto de datos está formado por 79330 instancias y 32 atributos, ya que ya nos hemos quedado con los valores del hotel de ciudad.

11.2.1 Valores nulos

Al analizar el conjunto de datos se ha podido observar algunos valores que eran nulos, estos aparecen con el nombre "nan". Vamos a ver dónde encontrábamos estos valores faltantes con ayuda de la tabla 1:

Tabla 1 Valores nulos

Atributo	Nº valores nulos	% Valores nulos del atributo
children	4	0.005%
country	24	0.030%
agent	8131	10.24%
company	75641	95.34%

Podemos observar, que de los 32 atributos que tenemos, solo en 4 aparecen valores nulos. En alguno de estos, los valores nulos eran muy pocos (children y country), pero en otros, como en agent o company, la cantidad de valores nulos era mucho mayor. A continuación, se detalla la manera en la que se ha procedido para eliminar este problema:

- **Children**: se ha decidido eliminar las instancias que tenían valores nulos, al ser tan pocas, solo 4, el impacto sobre el conjunto de datos es nulo.
- **Country**: se ha decidido eliminar las instancias que tenían valores nulos, al ser tan pocas, solo 24, el impacto sobre el conjunto de datos es nulo.

- **Agent**: para estos valores nulos, también se ha decidido eliminar las instancias que eran nulas. De esta manera, hemos reducido el conjunto de datos un 10%.
- **Company**: para este atributo, podemos ver que solo tenemos datos del 5% de las instancias. Por este motivo, se ha decidido eliminar todo el atributo completo.

Tras solucionar el problema de los datos nulos, el conjunto de datos contaba con 71181 instancias y con 31 atributos.

11.2.2 Limpieza del data set

Tras haber eliminado los datos nulos, es turno de observar los datos de cada atributo en busca de posible datos incorrectos o instancias que no tienen sentido. Los datos que se han observado han sido:

- **Meal**: los valores que toma la variable meal son 'BB', 'FB', 'HB', 'SC y 'Undefined'. 'BB' significa bed and breakfast, 'FB' full board (desayuno, comida y cena), y 'HB' half board (desayuno y comida o cena), mientras que 'SC' significa no definido, lo mismo que 'Undefined'. Esto nos puede traer problemas en el futuro, así que sustituiremos todos los valores 'Undefined' por 'SC'.
- **Adults**: se ha observado que hay instancias que están formados únicamente por niños y bebes. Este caso lo hemos considerado incorrecto, ya que consideramos que una reserva siempre debe tener un adulto mínimo. Se eliminan todas las instancias que tienen 0 adultos. Hay 332 instancias con 0 adultos.
- **Stays_in_week_nights & stays_in_weekend_nights**: estos atributos pueden tomar el valor 0, pero lo que no tiene mucho sentido es que ambos tomen el valor 0 de forma simultánea, ya que se trataría de una reserva sin número de noches. Por ese motivo eliminaremos todas las instancias que tengan valor 0 en ambos.
- **Adr**: recordemos que este atributo significa “average daily rate”, es decir, la tarifa diaria promedio de cada reserva. Hemos visto que hay instancias con valor adr = 0, la cual cosa tampoco tiene mucho sentido. En consecuencia, eliminaremos todas las instancias que tengan valor 0 para este atributo.
  En este atributo, tal y como se muestra en la ilustración 23, también se ha encontrado un valor muy anómalo:

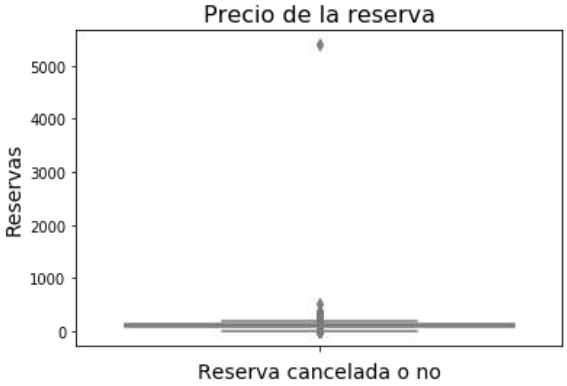


Ilustración 23 Anomalía en Precio de la reserva

Todos los adr están alrededor de 100 o 500 máximo, en cambio, hay un valor que sobrepasa los 5000. Se ha decidido eliminar dicho atributo.

Tras realizare estas modificaciones en el conjunto de datos, se ha reducido levemente la cantidad de datos, quedan un total 70349 instancias y 31 atributos.

11.2.3 Eliminación de atributos

En este apartado se han eliminado los atributos que nos iban a hacer servir en la aplicación de los algoritmos para conseguir el objetivo de predecir si una reserva iba al ser cancelada o no cuando esta era notificada al hotel.

Se han eliminado atributos que no aportaban nada al algoritmo:

- **hotel:** al trabajar únicamente los datos del hotel de ciudad, todas las instancias tienen el mismo valor en el atributo hotel. Por este motivo se ha decido eliminar este atributo.
- **arrival_date_year:** este atributo no nos servirá de nada si queremos predecir nuevos datos del futuro. Lo eliminamos.
- **deposit_type:** este atributo ha sido muy especial. A la hora de estudiarlo, se ha observado la ilustración 24, en la que se observa el numero de reservas en función del tipo de depósito.

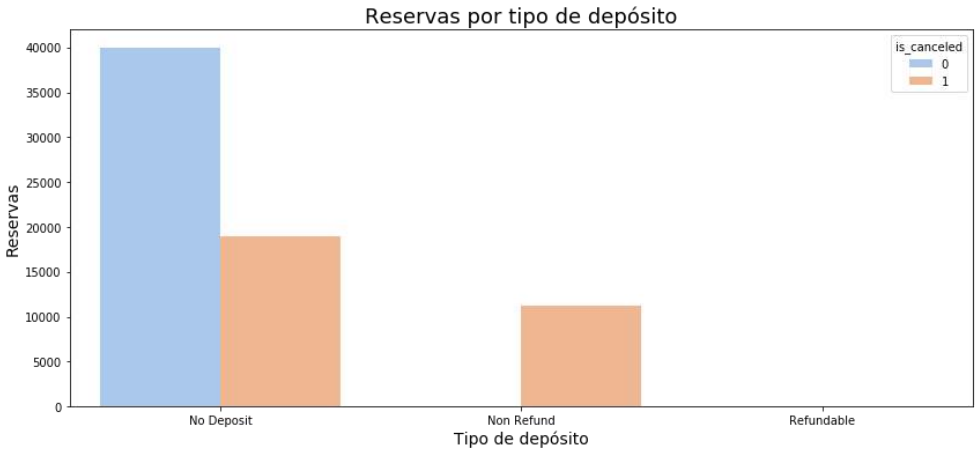


Ilustración 24 Reservas por tipo de atributo

Podemos ver que aproximadamente 20000 reservas sin depósito fueron canceladas, una cantidad realmente significativa que implicaría grandes pérdidas.

Sorprendentemente los depósitos no reembolsables tuvieron más cancelaciones que los reembolsables, ya que lo más lógico sería asumir el suceso contrario. Para ver este suceso más en detalle mostraremos un gráfico que solo refleje las reservas canceladas para cada tipo de depósito. Los resultados se pueden observar en la siguiente ilustración 25:

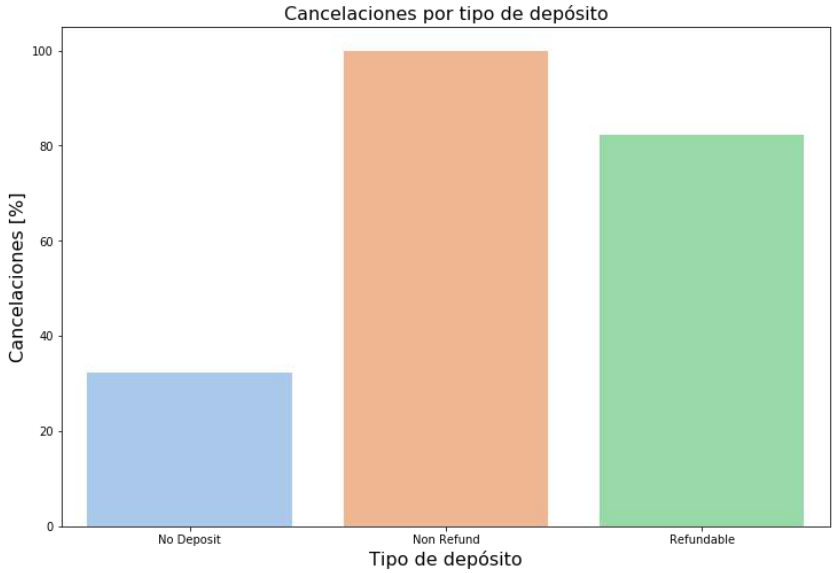


Ilustración 25 % de cancelaciones por tipo de deposito

Podemos ver que prácticamente la totalidad de las reservas con depósito no reembolsable fueron canceladas, lo cual no tiene sentido y nos lleva a preguntarnos si hay algún tipo de error en los datos. Como está muy claro que esto no es coherente procederemos a eliminar este atributo de nuestro conjunto de datos.

Recordamos que el objetivo es predecir si una reserva será cancelada o no cuando esta llega al hotel por primera vez. Al haber definido el objetivo de esta manera, se deben eliminar todos los atributos que van variando con el transcurso del tiempo hasta que la reserva se cancela o hasta que llega la fecha de llegada:

- **reservation_status:** este atributo que nos proporciona la misma información que la variable respuesta "is_canceled". "Reservation_status" nos indica si una reserva ha sido cancelada, si se hizo el check out, o si los huéspedes no se presentaron (no show). De esta manera, al estar proporcionándonos la misma información que la variable respuesta debemos eliminarla de nuestro dataset, ya que, sino sufriríamos problemas de "data leakage" ya que sino acabaríamos creando modelos demasiado optimistas que serían prácticamente inútiles en la realidad. Además, este atributo, nunca lo tendremos cuando la reserva llegue por primera vez al hotel.
- **booking_changes:** El atributo booking_changes lo eliminaremos ya que nuestro modelo se usaría cuando se realizase una reserva, y este atributo cambia en el tiempo, por lo que no nos aportarían información útil al recibir la reserva.
- **days_in_waiting_list:** lo eliminaremos ya que, al igual que los atributos anteriores, nuestro modelo se usaría cuando se realizase una reserva, y este atributo cambia en el tiempo, por lo que no nos aportarían información útil al recibir la reserva.
- **assigned_room_type**: es un atributo que es dependiente del tiempo, y en consecuencia irían cambiando con su paso. Además, no tendríamos información de este atributo una vez recibimos una reserva. Ya que la habitación se asignará cuando queden escasos días para la llegada del cliente.

- **reservation_status_date**: este valor nos marca en qué fecha se actualizado el "reservation_status" como es una medida que va cambiando con el tiempo y que no tendremos cuando la reserva llegue al hotel, la eliminamos de nuestro dataset.

11.3 DATA SET FINAL

Tras eliminar todas las instancias y los atributos que se han detallado en los apartados anteriores, se ha reducido el conjunto de datos hasta tener 70349 instancias y un total de 23 atributos. Los atributos que forman nuestro conjunto de datos final son:

- is_canceled
- lead_time
- arrival_date_month
- arrival_date_week_number
- arrival_date_day_of_month
- stays_in_weekend_nights
- stays_in_week_nights
- Adults
- children
- babies
- meal
- country
- market_segment
- distribution_channel
- is_repeated_guest
- previous_cancellations
- previous_bookings_not_canceled
- reserved_room_type
- agent
- customer_type
- adr
- required_car_parking_spaces
- total_of_special_requests

La variable a predecir será “is_canceled”, y los otros 22 atributos serán los que se usarán como predictores de esta. A continuación, en la tabla 2, se muestra el resumen de cada uno de los atributos numéricos:

Tabla 2 Resumen atributos numéricos

	count	mean	std	min	25%	50%	75%	max
is_canceled	70349.0	0.431278	0.495258	0.0	0.0	0.0	1.0	1.0
lead_time	70349.0	117.340289	112.578685	0.0	30.0	83.0	173.0	629.0
arrival_date_week_number	70349.0	27.300971	13.177260	1.0	17.0	27.0	38.0	53.0
arrival_date_day_of_month	70349.0	15.838164	8.751046	1.0	8.0	16.0	23.0	31.0
stays_in_weekend_nights	70349.0	0.824759	0.881099	0.0	0.0	1.0	2.0	10.0
stays_in_week_nights	70349.0	2.239293	1.411596	0.0	1.0	2.0	3.0	22.0
adults	70349.0	1.904405	0.468883	1.0	2.0	2.0	2.0	4.0
children	70349.0	0.090108	0.366083	0.0	0.0	0.0	0.0	3.0
babies	70349.0	0.004506	0.083081	0.0	0.0	0.0	0.0	10.0
is_repeated_guest	70349.0	0.007960	0.088865	0.0	0.0	0.0	0.0	1.0
previous_cancellations	70349.0	0.073633	0.354138	0.0	0.0	0.0	0.0	21.0
previous_bookings_not_canceled	70349.0	0.012139	0.368209	0.0	0.0	0.0	0.0	50.0
agent	70349.0	28.138893	56.429841	1.0	9.0	9.0	17.0	509.0
adr	70349.0	107.631035	37.429916	0.5	80.0	100.3	126.9	510.0
required_car_parking_spaces	70349.0	0.020953	0.144117	0.0	0.0	0.0	0.0	3.0
total_of_special_requests	70349.0	0.567030	0.784320	0.0	0.0	0.0	1.0	5.0

En esta tabla se puede ver cuántos valores hay en cada uno de los atributos, su media, la desviación típica, así como el valor mínimo y máximo, el primer cuartil, el segundo y tercero.

12 FAMILIARIZACIÓN DATASET

Con la finalidad de tener un mejor conocimiento sobre los datos con los que estamos tratando hemos creado una sección para analizarlos brevemente. De esta manera, crearemos algunos gráficos que nos ayudarán a entender mejor el contexto en el que estamos trabajando.

Esta sección se ha dividido en: el análisis de correlación de los atributos, información general de conjunto de datos y comparación de los atributos más importantes con la variable "is_canceled".

12.1 ANÁLISIS DE CORRELACIÓN

A continuación, se muestra la correlación que existe entre todas las variables numéricas de nuestro conjunto de datos. Hay un total de 15 atributos numéricos y se muestran en la ilustración 26:

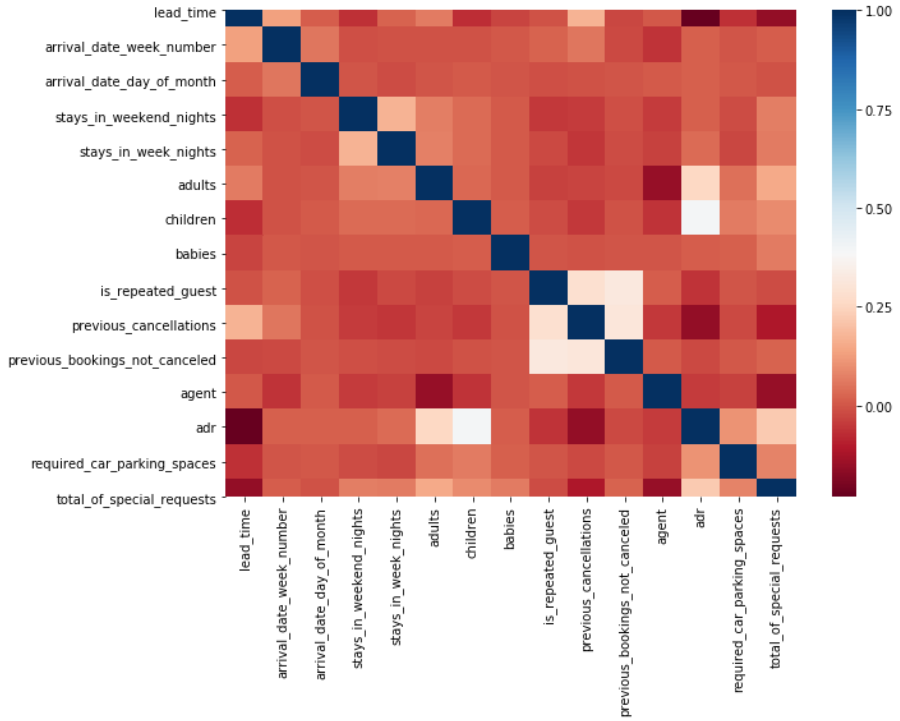


Ilustración 26 Correlación con todas las variables numéricas

No hay ninguna pareja de atributos numéricos que tengan una correlación superior a 0,5 en valor absoluto. De esta manera, se puede afirmar que nuestro conjunto de datos numéricos tiene una correlación baja.

Otro valor útil a mostrar es la correlación de las variables numéricas con la variable respuesta "is_canceled" que se muestra en la ilustración 27 de la siguiente página. Para entender e interpretar mejor el siguiente grafico se debe recordar que la variable "is_canceled" es binaria:

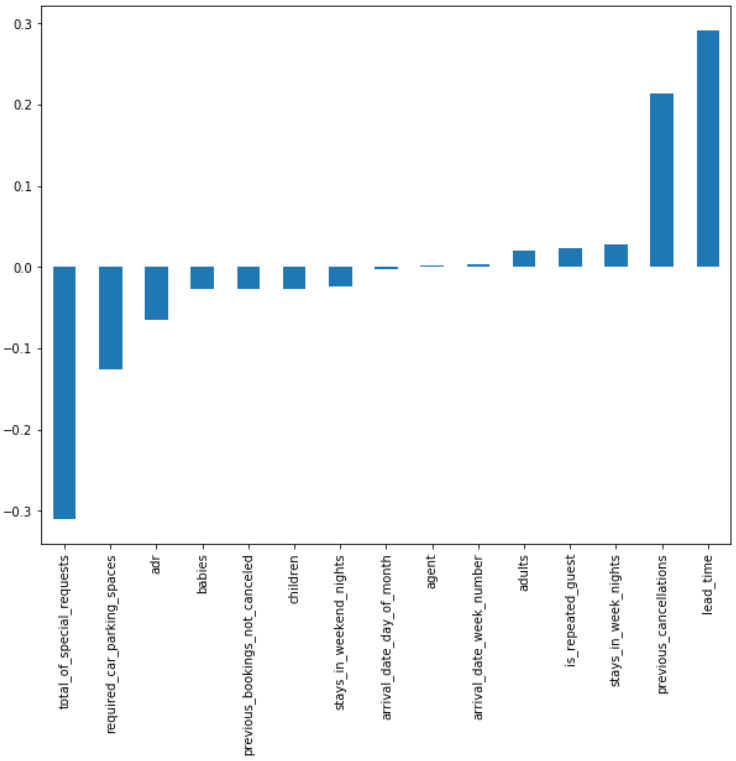


Ilustración 27 Correlación con las variables "is_canceled"

Podemos ver que las variables numéricas más correlacionadas con la variable respuesta son lead_time, previous cancellations, total_of_special_requestsy required_car_parking_spaces. La correlación de ninguno de estos atributos es superior a 0,3, por lo que también podemos afirmar que la correlación entre las variables numéricas y la variable binaria “is_cancelled” es muy pequeña.

12.2 INFORMACIÓN GENERAL

Antes de entrenar en más detalle con los datos, debemos contextualizar un poco. Por ejemplo, ¿cuánto vale una reserva por noche? ¿Hay meses del año más caros que otros? ¿Hay algún mes, en que hay muchas más reservas? En estos apartados se responderán estas y más preguntas que no ayudarán a entender mucho mejor los datos del hotel de la ciudad.

12.2.1 Información mensual

Unos de los factores más importantes de un hotel es la estacionalidad. Es decir, ¿hay meses en los que hay más clientes o menos? ¿Hay meses en los que se cancela mucho más? Para empezar, se muestra la ilustración 28 en la que se muestra para que meses se ha realizado la reserva:

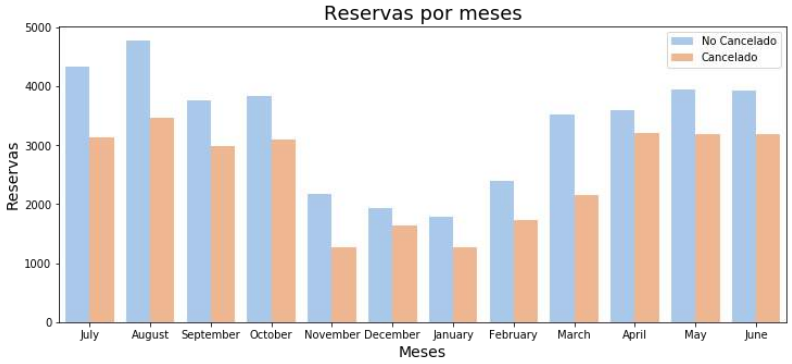


Ilustración 28 Reservas canceladas y no canceladas por meses

Parece ser que hay meses en la afluencia de persona en el hotel es más baja. Estos meses son los que coinciden en periodo navideño: noviembre, diciembre, enero y febrero. Pero, es importante destacar que, en algunos meses, solo hay datos de 2 años y en otros, de 3 años, por lo que es difícil de interpretar este gráfico. Para mejorarlos se grafican las reservas de meses por años:

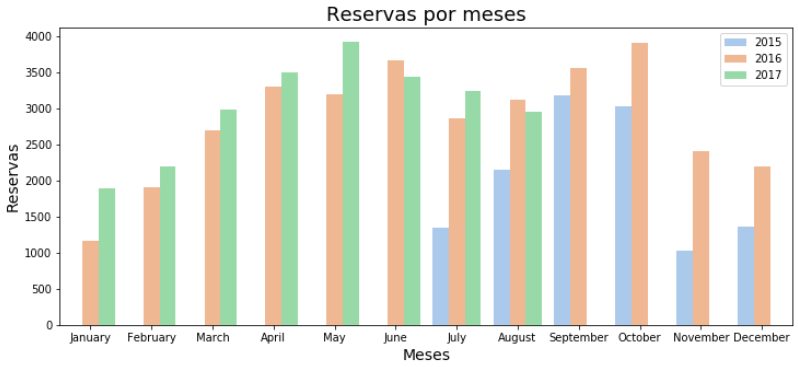


Ilustración 29 Reservas por meses por años

Para entender más la ocupación que hay en el hotel por meses, se ha graficado el número de reservas no canceladas medio de cada mes durante este periodo de 3 años en la ilustración 30, de esta manera, evitamos la problemática de tener más o menos datos de un mes:

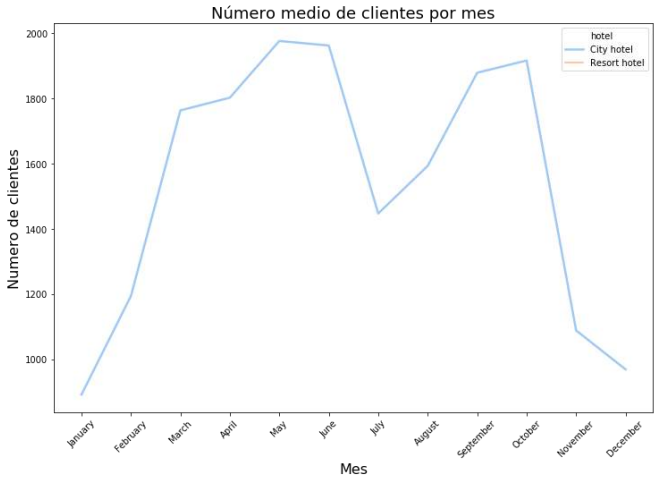


Ilustración 30 Número de clientes por meses

Con esta última ilustración 30, se puede confirmar que los meses de menos ocupación son los meses que coinciden con invierno y/o de navidad (noviembre, diciembre, enero y febrero). En cada uno de estos meses, pasan aproximadamente un total 1000 clientes por mes, muy lejos de los aproximadamente 1600 que pasan de media en cada uno de los meses restantes.

Otra observación apreciable es que hay un gran descenso de la ocupación en los meses de verano de julio y agosto. Este factor, se puede deber posiblemente, a la preferencia de los clientes por hoteles o resorts cercanos a la playa y piscina en tiempo de calor.

Durante los 3 años, el hotel ha tenido una afluencia media de 1.500 clientes por mes. Además, se ha calculado cual es el desembolso medio de una persona y por noche. El resultado obtenido ha sido: 59.3€.

Se ha observado que hay una estacionalidad razonable en cuanto a la cantidad de clientes por meses, pero, ¿hay algún tipo de estacionalidad en cuanto al tanto por ciento de reservas canceladas?:

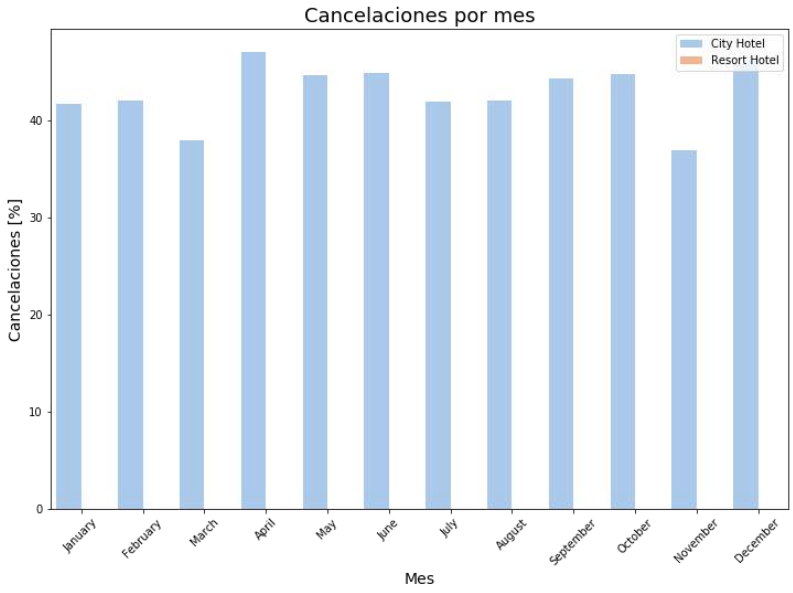


Ilustración 31 % de reservas canceladas por meses

Tal y como se aprecia en la ilustración 31 superior, el % de reservas canceladas se mantiene cercan al 40% durante todos los meses del año. Por lo que se puede decir que no hay una gran influencia del mes de la reserva con si la reserva fue cancelada o no.

12.2.2 Otra información

Una vez se tienen clara la cual es la importancia de los meses en el conjunto de datos, procederemos a estudiar cuantos días suelen reservas los clientes, y, si, en caso de cancelación de reserva, con cuantos días de antelación se avisa.

En concreto, en la siguiente ilustración 32, se muestra cual es la estancia más habitual de los clientes del hotel de ciudad:

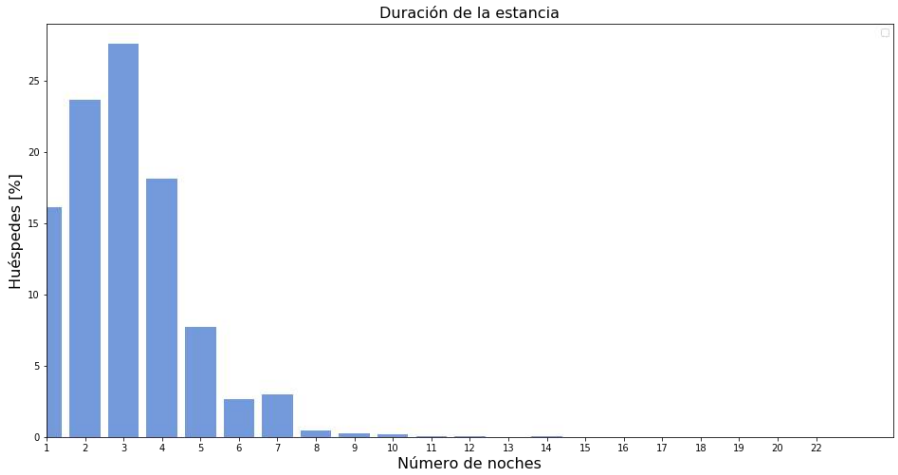


Ilustración 32 Duración de la estancia en días

Se puede ver que la mayoría de las reservas están formadas por 1, 2, 3 o 4 noches. En menor frecuencia se reservan 5, 6 o 7 noches. Es posible que haya algún tipo de oferta para una semana entera (7 días). Los otros números de noches se han repetido muy poco en nuestro conjunto de datos. Las reservas entre 1 y 7 días, ambos incluidos, representan el 99% de las reservas.

Otra información muy importante a conocer es la antelación con la que los clientes cancelan las reservas. Los resultados que hemos obtenido han sido los ilustrados en la figura 33:

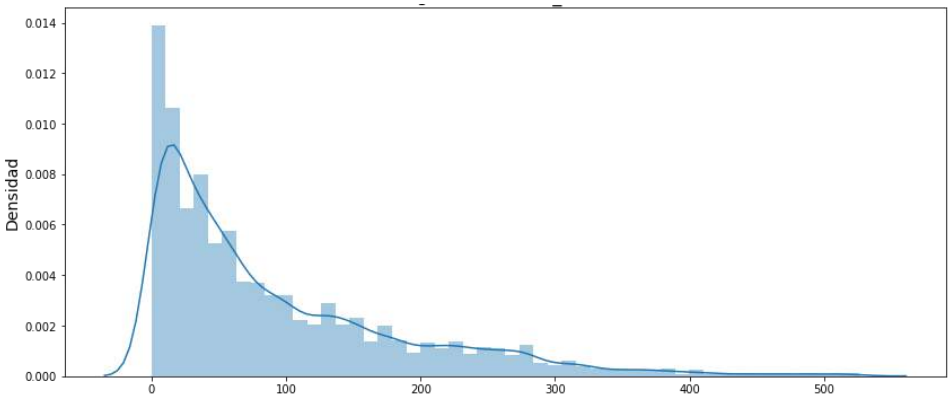


Ilustración 33 Antelación de días al cancelar una reserva

Podemos ver que cuanta menos antelación más cancelaciones tenemos, y eso es exactamente lo opuesto de lo que a los hoteles les interesa, sin duda un motivo de peso para justificar la creación de un modelo predictivo para poder prever estas cancelaciones y actuar en consecuencia a priori.

Se ha calculado que el 14,61% de las reservas canceladas se han cancelado con 10 o menos días de antelación.

Suponiendo que todas las reservas que se cancelaron con 10 días o menos de antelación no pudieron ser remplazables por otros clientes. Los ingresos que dejaría de percibir el hotel en los 3 años fueron (usando el valor medio de precio de estancia) de: 765632.36€. Es decir, estas reservas canceladas provocan dejar de ingresar unos 250.000€ al año.

De esta manera, podemos ver de primera mano los enormes beneficios que un modelo predictivo de las cancelaciones podría llegar a suponer para los responsables de los hoteles, ya que podrían modular su estrategia de ventas en consecuencia y amortiguar así el efecto de las elevadas cancelaciones, maximizando a la vez sus ingresos de manera considerable.

12.3 COMPARACIÓN DE ATRIBUTOS CON LA VARIABLE "IS_CANCELED"

En este apartado se graficarán las posibles importancias que puede haber entre algunos de los atributos más destacados y el atributo respuesta "is_canceled". Los atributos que se evaluaran son: lead_time, adr, country y total_of_special_request.

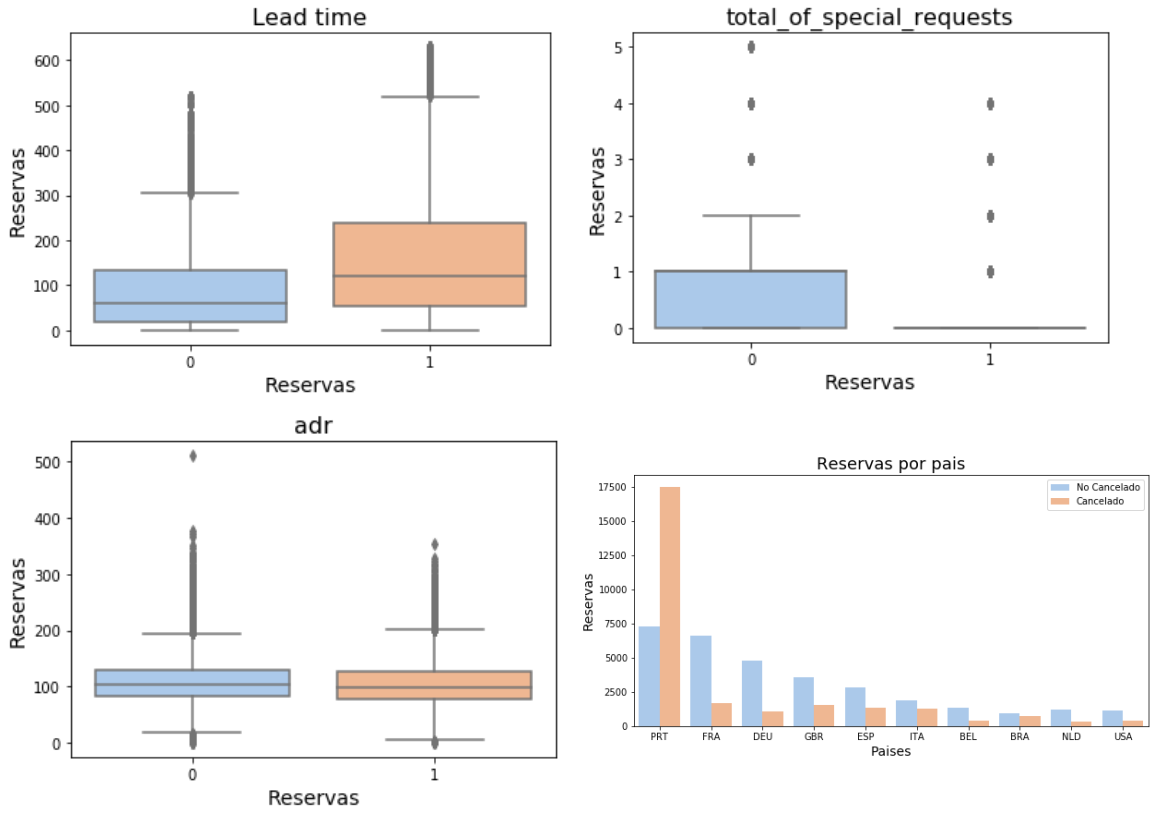


Ilustración 34 Comparación de los atributos lead_time, adr, country y total_of_special_request con la variable "is_canceled"

En los gráficos de la ilustración 34, se observa claramente que hay una cierta correlación con la antelación que se reserva. Las reservas canceladas suelen reservarse con una antelación superior a las no canceladas. Se Observa algo similar con el número de selección especial, las reservas no canceladas, tienen más requisitos en la reserva, esto atributo parece ser que denota cierto interés del cliente en la reserva. En cambio, el atributo que mide el precio por noche de cada reserva ("adr") no parece ser diferente entre las reservas cancelada o no. Todas estas afirmaciones se pueden comprobar también con la tabla número 3 de la siguiente página, en la que se muestra el valor medio de estas variables en función de sus variables de salida.

Tabla 3 Comparación de los atributos lead_time, adr y total_of_special_request con la variable "is_canceled"

is_canceled	lead_time	total_of_special_requests	adr
0	88.710265	0.779000	109.739788
1	155.094364	0.287508	104.850248

Cuando se compara si una reserva fue cancelada o no respecto a los países, se observa claramente que Portugal es un caso peculiar. Una reserva de clientes portugueses es mucho más probable que se cancele a que siga adelante. En concreto, una reserva portuguesa será cancelada el 70% de las veces, y, por el contrario, será únicamente convertida el 30% de las veces. Este comportamiento podría ser debido a diferencias culturales.

13 ¿COMO SE EVALUARÁN LOS MODELOS?

Una de las partes más importantes del aprendizaje supervisado es comparar los diferentes modelos que se entrenan. En este apartado se presentará la principal técnica para evaluar las prestaciones de cada uno de los modelos y las métricas que se utilizaran para comparar los modelos.

13.1 K VALIDACIÓN CRUZADA

La validación cruzada es un método estadístico utilizado para estimar la habilidad de los modelos de aprendizaje automático.

Se usa comúnmente en el aprendizaje automático aplicado para comparar y seleccionar un modelo para un problema de modelado predictivo dado porque es fácil de entender, fácil de implementar y da como resultado estimaciones de habilidades que generalmente tienen un sesgo menor que otros métodos.

En concreto, la validación cruzada es un procedimiento de re muestreo utilizado para evaluar modelos de aprendizaje automático en una muestra de datos limitada.

El procedimiento tiene un único parámetro llamado k que se refiere al número de grupos en los que se dividirá una muestra de datos determinada. Como tal, el procedimiento a menudo se llama validación cruzada k-fold. Cuando se elige un valor específico para k, se puede usar en lugar de k en la referencia al modelo, como k = 10 convirtiéndose en una validación cruzada de 10 veces. El proceso de validación cruzada es repetido durante k iteraciones, con cada uno de los posibles subconjuntos de datos de prueba. Finalmente se realiza la media aritmética de los resultados de cada iteración para obtener un único resultado.

Ilustración 35 K-fold

La validación cruzada usa una muestra limitada de datos para estimar cómo se espera que el modelo funcione en general cuando se usa para hacer predicciones sobre datos que no se usaron durante el entrenamiento del modelo.

Suponemos que tenemos un modelo con uno o más parámetros de ajuste desconocidos y unos datos de entrenamiento que queremos analizar. El proceso de ajuste optimiza los parámetros del

modelo para que éste se ajuste a los datos de entrenamiento tan bien como pueda. Si tomamos una muestra independiente como dato de prueba (validación), del mismo grupo que los datos de entrenamiento, normalmente el modelo no se ajustará a los datos de prueba igual de bien que a los datos de entrenamiento. Esto se denomina sobreajuste y acostumbra a pasar cuando el tamaño de los datos de entrenamiento es pequeño o cuando el número de parámetros del modelo es grande. La validación cruzada es una manera de predecir el ajuste de un modelo a un hipotético conjunto de datos de prueba cuando no disponemos del conjunto explícito de datos de prueba.

13.2 MÉTRICAS

13.2.1 Matriz de confusión

Para entender las métricas que se utilizarán para evaluar los modelos, primero se debe definir y entender qué es una matriz de confusión.

Para evaluar un modelo que hemos creado, podríamos simplemente calcular su precisión ("accuracy"), como la proporción entre las predicciones correctas que ha hecho el modelo y el total de predicciones. Sin embargo, aunque en ocasiones resulta práctico por su facilidad de cálculo, otras veces es necesario profundizar un poco más y tener en cuenta los tipos de predicciones correctas e incorrectas que realiza el clasificador. Es aquí donde entra en juego la Matriz de Confusión.

La matriz de confusión se define como una herramienta que permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real. Uno de los beneficios de las matrices de confusión es que facilitan ver si el sistema está confundiendo las diferentes clases o resultados de la clasificación.

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Ilustración 36 Matriz de confusión

- TP es la cantidad de positivos que fueron clasificados correctamente como positivos por el modelo.
- TN es la cantidad de negativos que fueron clasificados correctamente como negativos por el modelo.
- FN es la cantidad de positivos que fueron clasificados incorrectamente como negativos. Error tipo 2 (Falsos Negativos)
- FP es la cantidad de negativos que fueron clasificados incorrectamente como positivos. Error tipo 1 (Falsos positivos)

13.2.2 Precisión y Exactitud

Una vez se conocen los elementos que forman la matriz de confusión, procedemos a definir que métricas básicas se pueden extraer de estas. Las principales métricas de la matriz de confusión y más importantes son la precisión, la precisión balanceada y la exactitud. Se definen a continuación:

- **Precision "Accuracy":** se refiere a lo cerca que está el resultado de una medición del valor verdadero. En términos estadísticos, la exactitud está relacionada con el sesgo de una estimación. También se conoce como Verdadero Positivo (o "True positive rate"). Se representa por la proporción entre los positivos reales predichos por el algoritmo y todos los casos positivos. La fórmula es la siguiente:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

- **Precision balancead "Balanced Accuracy":** es una métrica que se puede usar para evaluar qué tan bueno es un clasificador binario. Es especialmente útil cuando las clases no están balanceadas. Se calcula como el promedio de la proporción de valores correctos de cada clase de manera individual:

$$Balanced\ Accuracy = \frac{TP/(TP + FN) + \ TN/(TN + FP)}{2} = \frac{Sensibilidad + Especificidad}{2}$$

- **Exactitud "Precision":** Se refiere a la dispersión del conjunto de valores obtenidos a partir de mediciones repetidas de una magnitud. Cuanto menor es la dispersión mayor la precisión. Se representa por la proporción entre el número de predicciones correctas (tanto positivas como negativas) y el total de predicciones. También se conoce como Verdadero Positivo (o "True positive rate"). La fórmula es:

$$Precision = \frac{TP}{TP + FP}$$

La siguiente ilustración ayuda a ver de forma práctica la diferencia entre precisión y exactitud. Así, por ejemplo, la figura (b) representa un resultado exacto y preciso, mientras que la (C) es preciso, pero no exacto y la (a) no es ni una cosa ni la otra:

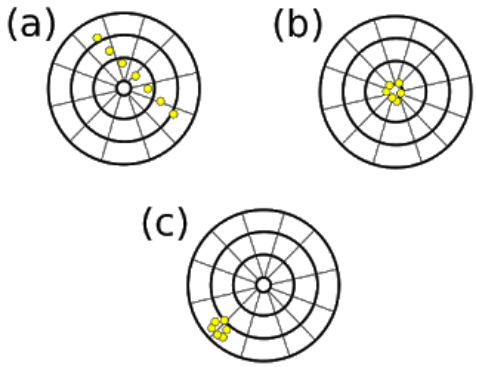


Ilustración 37 Diferencia entre precisión y exactitud

13.2.3 Sensibilidad y Especificidad

A parte de la precisión y la exactitud también se puede obtener la sensibilidad y la especificidad. Estos son dos valores que nos indican la capacidad del estimador para discriminar los casos positivos, de los negativos. La sensibilidad es la fracción de verdaderos positivos, mientras que la especificidad, es la fracción de verdaderos negativos.

- **Sensibilidad o exhaustividad ("Recall"):** también se conoce como Tasa de Verdaderos Positivos (True Positive Rate) (TP). Es la proporción de casos positivos (en nuestro ejemplo, setas venenosas) que fueron correctamente identificadas por el algoritmo. Se calcula según la ecuación:

$$Sensibilidad = \frac{TP}{TP + FN}$$

- **Especificidad**: por otra parte, es la Tasa de Verdaderos Negativos, ("true negative rate" o TN). Se trata de los casos negativos que el algoritmo ha clasificado correctamente. Su ecuación es:

$$Especificidad = \frac{TN}{TN + FP}$$

13.2.4 F1 Score

El valor F1 se utiliza para combinar las medidas de precision y recall en un solo valor. Esto es práctico porque hace más fácil el poder comparar el rendimiento combinado de la precisión y la exhaustividad entre varias soluciones. Es de gran utilidad cuando la distribución de las clases es desigual. La fórmula es:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

13.2.5 Roc Curve

La curva ROC (acrónimo de Receiver Operating Characteristic, o Característica Operativa del Receptor) es una medida de rendimiento para problemas de clasificación en varios ajustes de umbrales. ROC es una curva de probabilidad y AUC representa el grado o medida de separabilidad. Indica cuánto es capaz un modelo de distinguir entre diferentes clases. Cuanto mayor sea el AUC, valor, mejor será el modelo para predecir 0s como 0s y 1s como 1s. Es decir, cuanto mayor sea el AUC, mejor será el modelo para distinguir entre clases. Esta curva representa dos parámetros:

- Tasa de verdaderos positivos (TPR)
- Tasa de falsos positivos (FPR)

Tasa de verdaderos positivos (TPR) es sinónimo de exhaustividad, valor definido previamente en el apartado de matriz de confusión.

La tasa de falsos positivos (FPR), no se ha había visto anteriormente, esta se define de la siguiente manera:

$$FPR = \frac{FP}{FP + VN}$$

Una curva ROC representa TPR frente a FPR en diferentes umbrales de clasificación. Reducir el umbral de clasificación clasifica más elementos como positivos, por lo que aumentarán tanto los falsos positivos como los verdaderos positivos. Dado que VPR es equivalente a sensibilidad y FPR es igual a 1-especificidad, el gráfico ROC también es conocido como la representación de sensibilidad frente a (1-especificidad). En la siguiente ilustración 38, se muestra dos curvas ROC típicas:

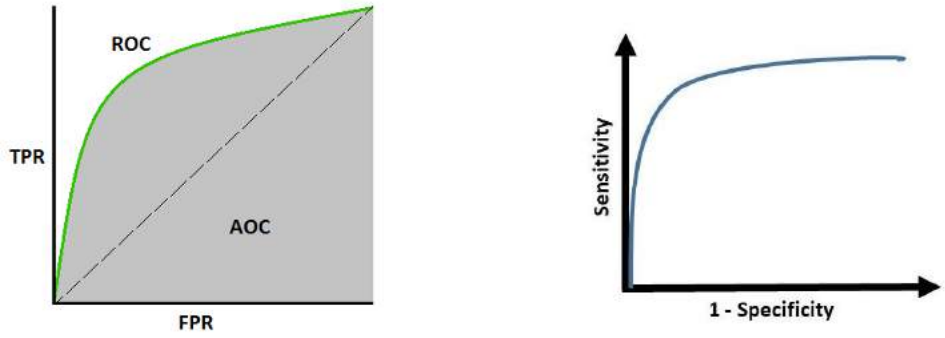


Ilustración 38 Curvas ROC

AUC es un algoritmo eficiente basado en ordenamiento que permite evaluar un modelo de muchas veces con diferentes umbrales de clasificación. AUC significa "área bajo la curva ROC". Esto significa que el AUC mide toda el área bidimensional por debajo de la curva ROC completa.

El AUC proporciona una medición agregada del rendimiento en todos los umbrales de clasificación posibles. Una forma de interpretar el AUC es como la probabilidad de que el modelo clasifique un ejemplo positivo aleatorio más alto que un ejemplo negativo aleatorio. A modo de ejemplo, la siguiente ilustración 39, muestra datos que están ordenados de izquierda a derecha en orden ascendente con respecto a las predicciones de un modelo:

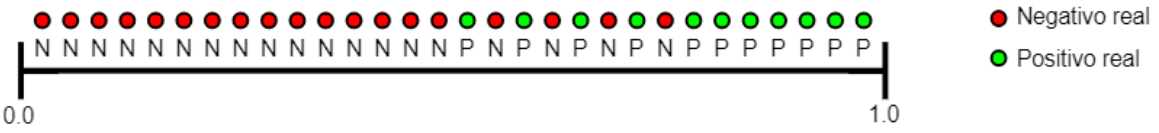


Ilustración 39 Interpretación AUC

El AUC representa la probabilidad de que un ejemplo aleatorio positivo (verde) se posicione a la derecha de un ejemplo aleatorio negativo (rojo).

El AUC oscila en valor del 0 al 1. Un modelo cuyas predicciones son un 100% incorrectas tiene un AUC de 0,0; otro cuyas predicciones son un 100% correctas tiene un AUC de 1,0.

El área bajo la curva Roc es conveniente por las dos razones siguientes:

- El AUC es invariable con respecto a la escala. Mide qué tan bien se clasifican las predicciones, en lugar de sus valores absolutos.
- El AUC es invariable con respecto al umbral de clasificación. Mide la calidad de las predicciones del modelo, sin tener en cuenta qué umbral de clasificación se elige.

14 ESTUDIO GENÉRICO INICIAL

En esta parte del trabajo se realizará un pequeño estudio de comparación de los principales algoritmos de clasificación en cuanto a prestaciones. Para poder compararlos, se utilizará el conjunto de datos de las reservas de hoteles.

Los modelos que se han decidido estudiar han sido: Naive Bayes, Logistic regression, Decision Tree, Random Forest, Bagging, AdaBoost y XGBoost.

Cada uno de estos modelos será entrenado con los hiperparámetros que ofrece el modelo por defecto, teniendo así, una visión más genérica de comportamiento del modelo. En los apartados siguientes, se ajustarán los hiperparámetros de algunos modelos para conseguir unas prestaciones más elevadas.

Es importante comentar que las variables categóricas han tenido que ser codificadas a numéricas con la técnica “one hot encoder”, la estrategia que implementa es crear una columna para cada valor distinto que exista en la característica que estamos codificando y, para cada registro, marcar con un 1 la columna a la que pertenezca dicho registro y dejar las demás con 0.

Para realizar este pequeño resumen de las prestaciones de los principales algoritmos de clasificación se ha utilizado la técnica de K cross validation con 10 capas con todos lo datos. De esta manera, cada algoritmo se entrena y se valida 10 veces con datos diferentes. Las métricas finales serán la media de los resultados de validación, con esto conseguimos unos resultados más reales y evitamos errores de tener los datos de entrenamiento y de test mal seleccionados. Además, evitamos el sobre ajuste en cada uno de los modelos.

Los resultados que hemos obtenido para cada uno de los modelos han sido los mostrados en la tabla 4 y en la ilustración siguiente:

Tabla 4 Resultados de los principales modelos con 10-cross validation y sin ningún ajuste

	Model	Accuracy	Balanced accuracy	ROC AUC	TN	FP	FN	TP
0	Naive Bayes	0.4707	0.5330	0.5349	321.0	3679.0	43.0	2990.0
1	Logistic regression	0.7933	0.7827	0.8704	3439.0	561.0	892.0	2141.0
2	Decision tree	0.8477	0.8450	0.8481	3458.0	542.0	529.0	2504.0
3	Random Forest	0.8799	0.8733	0.9535	3684.0	316.0	528.0	2505.0
4	Decision tree bagging	0.8707	0.8645	0.9423	3639.0	361.0	548.0	2485.0
5	AdaBoost	0.8088	0.7996	0.8931	3468.0	532.0	812.0	2221.0
6	XGBoost	0.8598	0.8537	0.9414	3594.0	406.0	579.0	2454.0

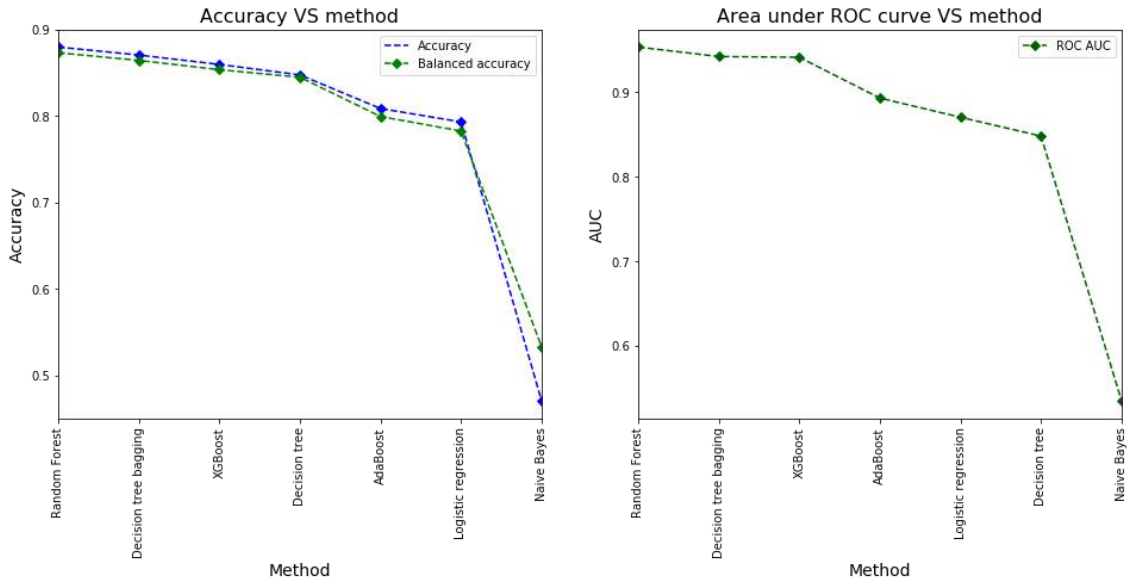


Ilustración 40 Accuracy y AUC de los principales modelos con 10-cross validation y sin ningún ajuste

Podemos ver, que los modelos que mejor se comportan con los datos que tenemos parecen ser el: Random Forest, Decision Tree Bagging y el XGboost. Todos ellos tienen un valor aproximados de acierto del 85%. El único algoritmo que consigue resultados muy inferiores comparaos con los otros es el Naive Bayes, que obtiene un acierto sobre el 50%.

En los siguientes apartados, se estudiarán cada uno de los algoritmos que se han descrito en la parte teórica por separado (Decision Tree, Random Forest, Bagging y Adaboos) y se modificarán y ajustarán algunos de sus parámetros para poder obtener una modelo final con mejores prestaciones.

15 MEJORANDO ÁRBOL DE DECISIÓN

En este apartado vamos a trabajar en concreto con el algoritmo de árbol de decisión. El objetivo es, cambiando algunos parámetros del modelo, encontrar el mejor algoritmo posible. Para ello, primero de todo, se reducirán los numero de atributos para reducir el coste computacional. El segundo paso será seleccionar los hiperparámetros que nos hagan tener un modelo más robusto.

Para realizar el estudio de la importancia de atributos hemos dividido el conjunto de datos en 20% de test, 20% de validación y 60% de entrenamiento.

Ilustración 41 Separación de los datos

Los datos de entrenamiento servirán para entrenar el modelo. Los datos de validación se usarán para ajustar los hiperparámetros del modelo y los datos de test se emplearán para obtener los resultados del modelo. Para evaluar el funcionamiento del algoritmo se usan datos que no han sido vistos por el modelo nunca, obteniendo así, una valoración más fiable y real.

15.1 SELECCIÓN DE ATRIBUTOS

Para seleccionar los atributos que eran más importantes en el árbol de decisión se ha utilizado la característica llamada importancia de atributos o características. La importancia de la característica se refiere a un tipo de técnica que se usa para asignar puntajes a las características de entrada de un modelo predictivo que indica la importancia relativa de cada característica al hacer una predicción.

Con los datos de entrenamiento, se ha realizado un 10-cross validation para obtener la importancia media de los atributos, teniendo así, un valor más real, seguro y evitando el sobre ajuste. Como ya se ha comentado, los datos de validación se utilizarán para ajustar los hiperparámetros. Los datos de test únicamente se usarán para hacer el test final del modelo, pero en ningún caso se utilizará para mejorar el modelo o ajustarlo.

Al empezar, el modelo inicial estaba formado por todos 22 atributos. Se ha aplicado la importancia de características y decidimos eliminar los 10 atributos que menos importaban en el modelo, quedándonos así con los 12 atributos más importantes. El siguiente paso fue sacar la importancia de características del modelo con 12 atributos, en este caso, se eliminaron los 8 atributos menos importantes y nos quedamos con un modelo de únicamente 4 variables. A continuación, mostramos los gráficos de la importancia de las características de cada uno de los modelos que se han estudiado:

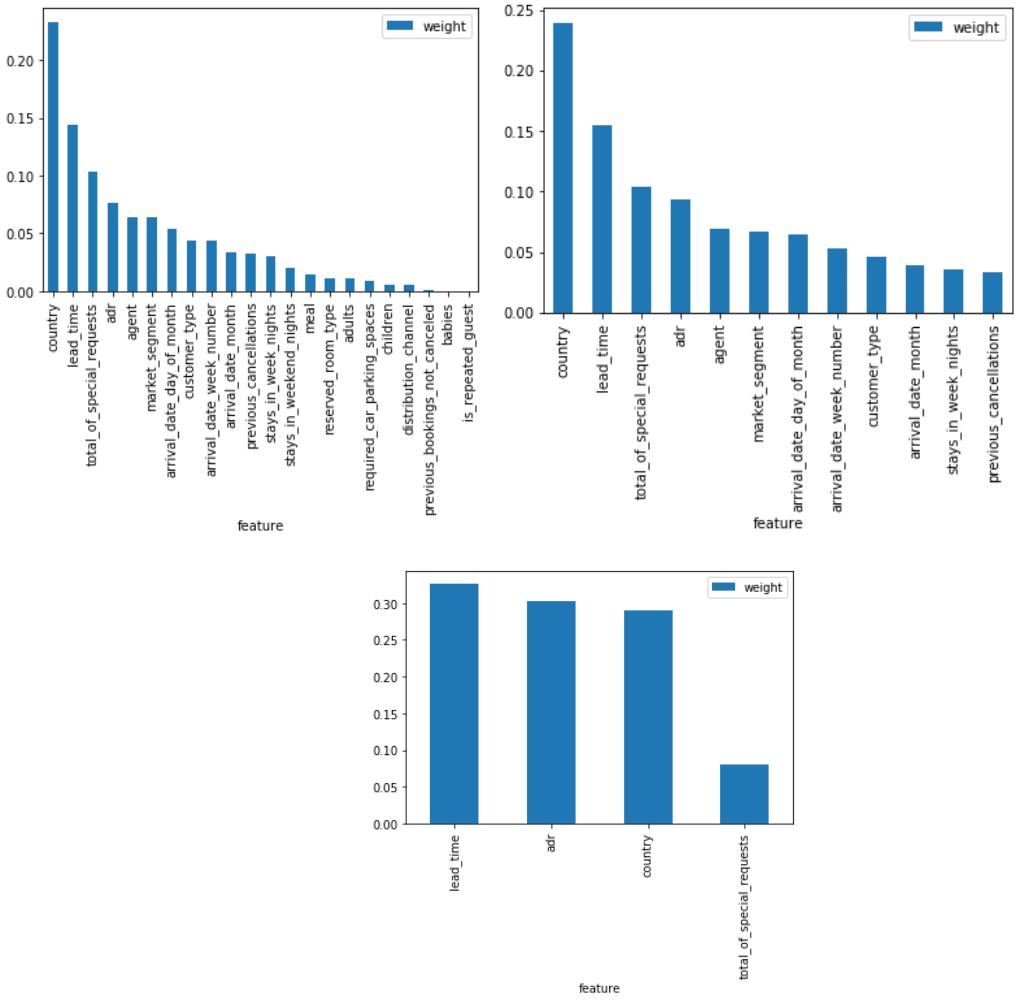


Ilustración 42 Importancia de atributos en árbol de decisión

Una vez se ha observado la justificación de la selección de los atributos, se muestra cuáles son las prestaciones de cada uno de los modelos. En la ilustración 43, se puede observar una comparación de la métrica "accuracy" y AUC de los tres modelos:

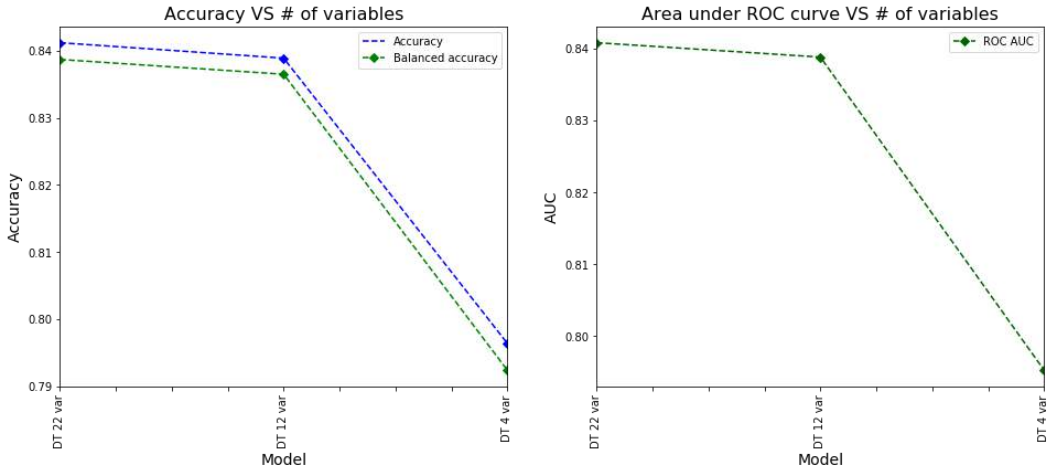


Ilustración 43 Accuracy y AUC por cada modelo

Se puede observar que el modelo de 22 variables y el de 12 tiene casi las mismas prestaciones. En cambio, el modelo de 4 variables se comporta un poco peor. Viendo estos gráficos se ha decidido quedarnos con el modelo de las 12 variables, ya que este mantiene casi los niveles de acierto iniciales y, además, se reducen mucho los tiempos computacionales.

Los 12 atributos que forman el modelo son:

- lead_time,
- arrival_date_week_number
- arrival_date_day_of_month
- stays_in_week_nights,
- previous_cancellations
- agent
- adr
- total_of_special_requests
- arrival_date_month
- country
- market_segment
- customer_type

15.2 AJUSTE DE HIPERPARÁMETROS

Una vez tenemos seleccionadas todas las variables seleccionadas para usar en nuestro modelo es momento de ajustar algunos parámetros del algoritmo de árbol de decisión. El algoritmo tiene muchos parámetros, todos los parámetros que están asignados para nuestro modelo son más que correctos y no serán modificados. El único que se modificará será el “max_depth”.

Este hiperparámetro hace referencia a la profundidad máxima que puede tener nuestro árbol. Por defecto el árbol es infinitamente largo hasta que se encuentra un nodo hoja. Este puede llevar a un sobre ajuste con los datos de entrenamiento. Por este motivo se ha decidido graficar como evolucionaba las prestaciones de nuestro modelo en función de los valores de “max_depth”. Para realizar este estudio se ha dividido otra vez el conjunto de datos de entrenamiento. Teniendo finalmente 60% de datos de entrenamiento, 20% de datos de validación y 20% de datos de test. Los datos de validación se utilizarán para poder ajustar este parámetro sin hacerlo directamente con los datos de test.

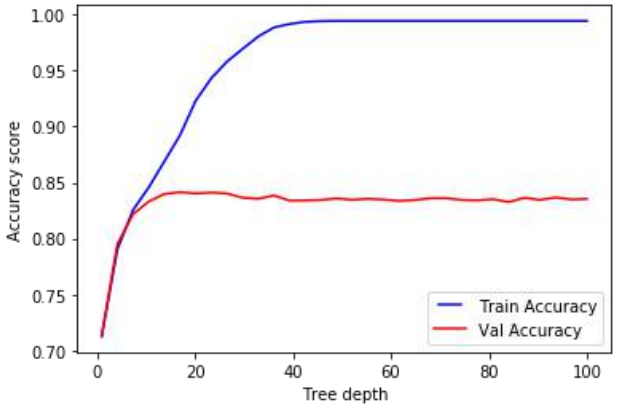


Ilustración 44 Ajuste de profundidad máxima

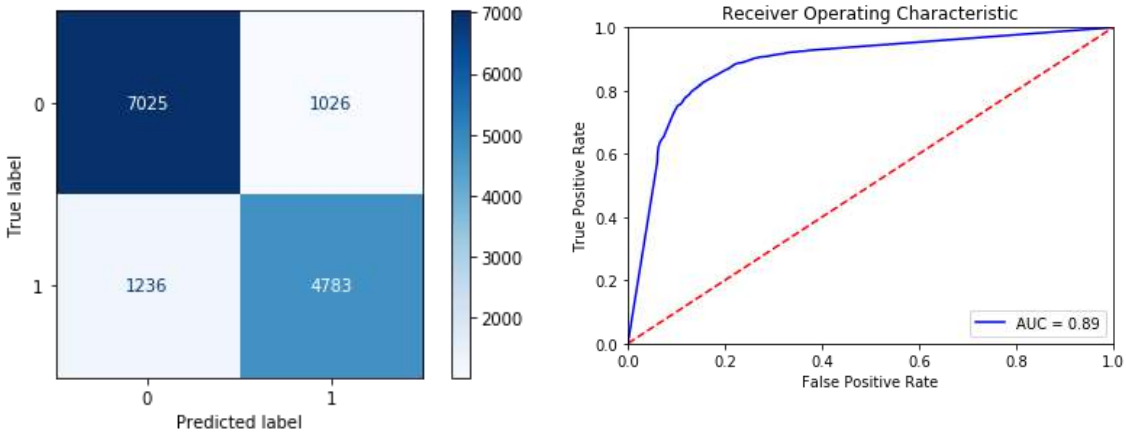
Gracias a la ilustración superior 44 podemos ver que con un valor superior a 20 de profundidad máxima del árbol el modelo tiende a sobre ajustarse. Por ese motivo hemos decidido quedarnos con el modelo que tiene una profundidad máxima de 20.

15.3 MODELO FINAL

Tras realizar la selección de atributos y el ajuste de hiperparámetros ya está todo listo para poder entrenar y obtener el modelo definitivo final. El modelo definido tiene los siguientes hiperparámetros:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=20, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Para entrenar dicho se utilizará los 12 atributos que se han seleccionado en el apartado anterior. Los datos que se utilizaran para entrenar el modelo han sido los de entrenamiento, que suponen el 60% de los datos. Una vez entrenando el modelo, se han calculado los resultados con los datos de test, los cuales no han sido vistos nunca por el modelo. Los resultados obtenidos han sido los observados en la ilustración 45:



Modelo	Accuracy	Bal. acc	Roc Auc	F1 Score	Sensitivity	Specificity
A. de decisión	0,839	0,833	0,886	0,808	0,80	0,87

Ilustración 45 Resultados finales árbol de decisión

El valor de Sensitivity hace referencia a la precisión que tiene el modelo acertando los casos que son de la clase 1, es decir, acertando reservas canceladas. En cambio, la Specificity es la precisión que tiene el modelo acertando la clase 0, es decir, las reservas canceladas. Ambos se calculan dividiendo el número de clase predicho correctamente por el número total de valores reales de esa clase. Con la matriz de confusión de arriba, la sensitivity se calculara 4783/(1236+4783)=0,8.

16 MEJORANDO ÁRBOL DE DECISIÓN BAGGING

En este apartado se trabajará en detalle sobre el comportamiento del algoritmo del árbol de decisión bagging. Al igual que en el apartado anterior del algoritmo de árbol de decisión, primero se intentarán reducir la cantidad de atributos para mejorar el coste computacional, y finalmente, se ajustarán los hiperparámetros, consiguiendo así, un modelo más acurado y robusto.

Se utilizarán las mismas proporciones de separación del conjunto de datos, es decir, 60% datos de entrenamiento, 20% de validación y 20% de test.

16.1 SELECCIÓN DE ATRIBUTOS

Para intentar reducir el coste computación del problema, se han seleccionado los atributos más importantes del modelo bagging de manera iterativa, consiguiendo así tener 3 modelos diferentes, formados por 22 atributos, 12 atributos y 4 atributos respectivamente. La importancia de los atributos se ha obtenido realizando un cross validadition de 10 capas de los datos de entrenamiento, teniendo así unos resultados más confiables. Con esta técnica, también se ha obtenido la efectividad del modelo. Las importancias de los atributos que se han conseguido han sido:

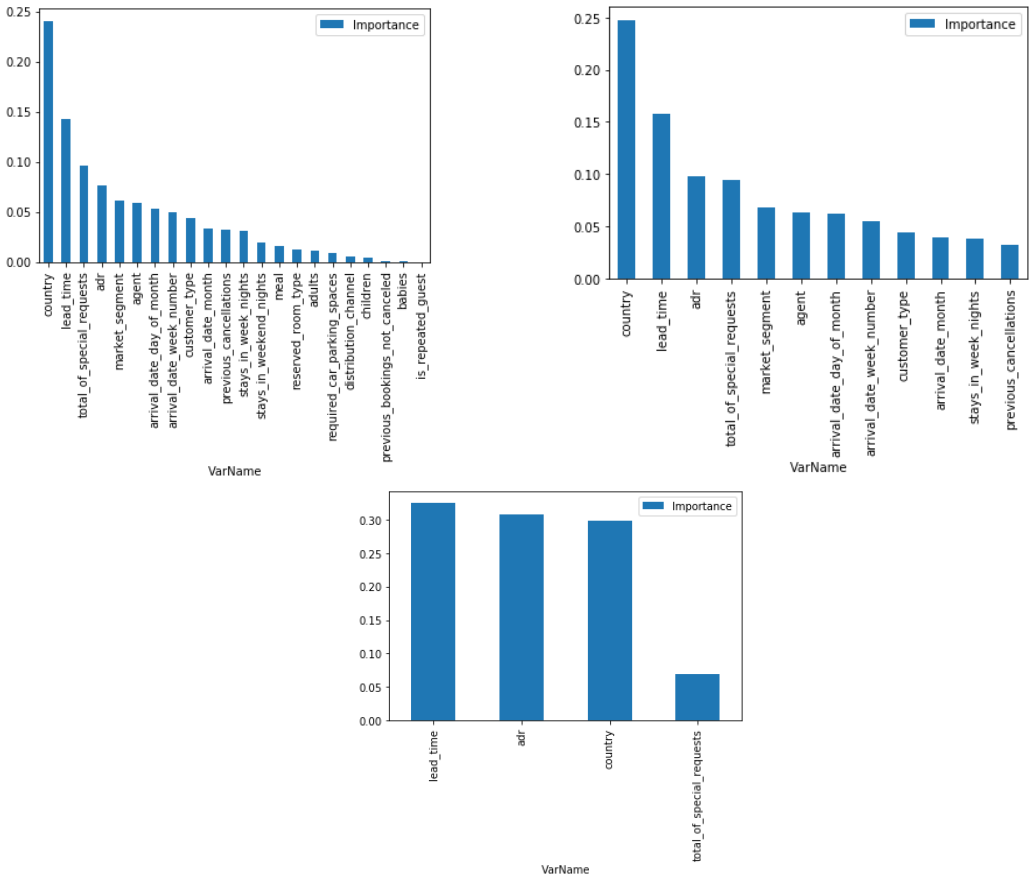


Ilustración 46 Importancia atributos bagging

Como se ha comentado, gracias a esta información de la importancia de los atributos se han generado 3 modelos. Uno con 22 atributos, el segundo con 12 atributos y el ultimo con solo 4 atributos. A continuación, se compara la efectividad de cada uno de los modelos:

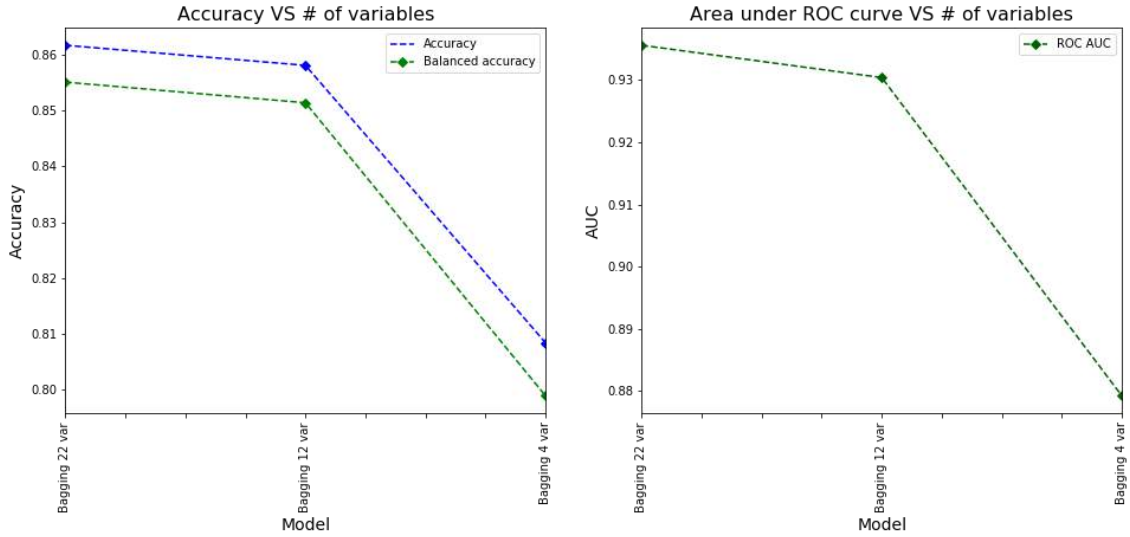


Ilustración 47 Accuracy y AUC por cada modelo

Se observa que los modelos de 22 atributos y 12 atributos se comportan de manera muy similar, teniendo una efectividad superior al 85%. En cambio, el modelo de únicamente de 4 variables, obtiene resultados más inferiores.

Se ha decidido trabajar con el modelo de 12 atributos ya que el nivel de prestaciones en cuanto a efectividad es muy similar al modelo inicial con 22 atributos, pero, además, este modelo reducido tiene un coste computacional mucho menor.

Los atributos que forman el modelo de 12 atributos son:

- lead_time,
- arrival_date_week_number
- arrival_date_day_of_month
- stays_in_week_nights,
- previous_cancellations
- agent
- adr
- total_of_special_requests
- arrival_date_month
- country
- market_segment
- customer_type

Cabe destacar que estos atributos son también los 12 más importantes del algoritmo del árbol de decisión. Pero, tal y como se puede ver en la ilustración 48 de la siguiente página, los valores de importancia no son los mismos entre los algoritmos de bagging y árbol de decisión.

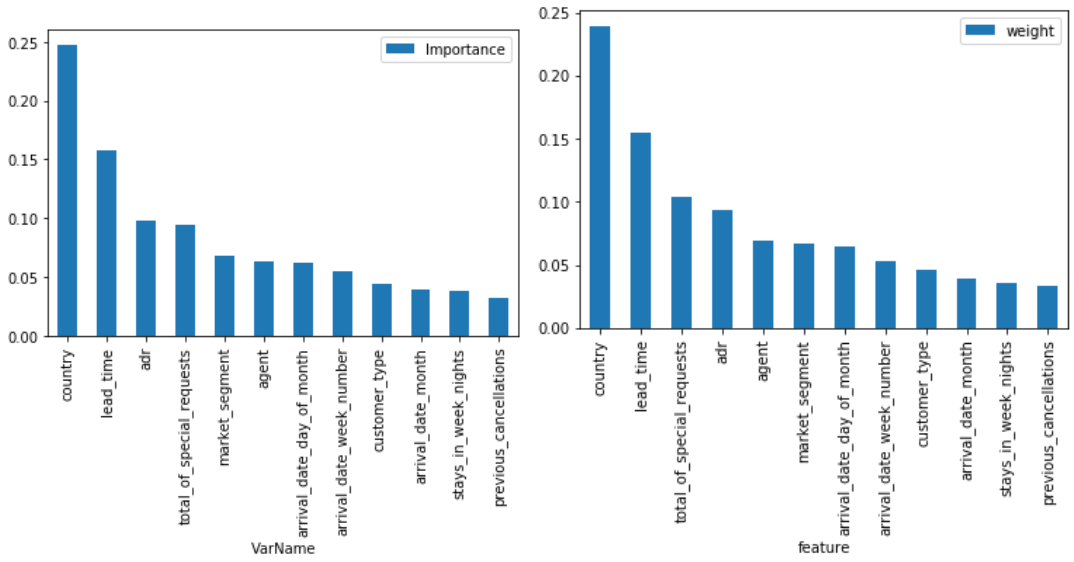


Ilustración 48 Comparación de importancia de atributos de los modelos bagging y árbol de decisión

16.2 AJUSTE DE HIPERPARÁMETROS

En este apartado se ajustarán algunos parámetros del algoritmo para poder conseguir unos resultados más acurados y ajustados a nuestro problema. El modelo será entrenando con los datos de entrenamiento y se observaran los resultados utilizando los datos de validación. Los parámetros que se han ajustado son: n_estimators y max_samples.

n_estimators

El primer hiperparámetro que se ajustará es el n_estimators. Este parámetro hace referencia al número de bolsas de datos que queremos se creen, y, en consecuencia, el número de árboles de decisión. Por defecto, el valor definido es 10, a continuación, en la ilustración 49, se muestra cómo evoluciona la prestación del modelo en función del n_estimators:

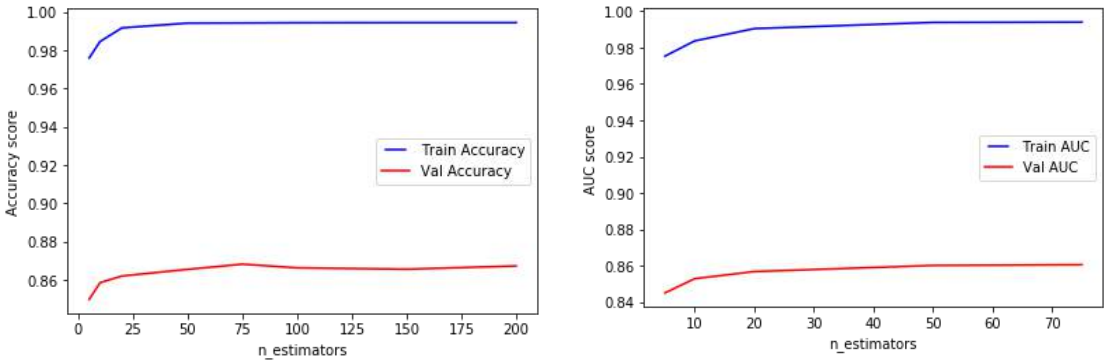


Ilustración 49 Ajuste número de estimadores

Con los datos que se observan, se ha decidido que el hiperparámetro n_estimators será ajustado a un valor de 25.

max_samples

Tras tener el n_estimators definido, se ha ajustado el parámetro max_samples. Este valor hace referencia a la cantidad que queremos por el cual este formado cada bolsa de datos del bagging. El valor es un valor del 0 al 1, y es la proporción de datos que formaran la bolsa en función de los datos totales de entrenamiento. Por ejemplo, si se disponen de un conjunto de datos de entrenamiento de 100 datos y el max_samples es de 0.4, cada una de las bolsas del bagging estará formada por 40 datos.

En la ilustración 50, se muestra la evolución de las prestaciones del modelo en función del número de max_samples:

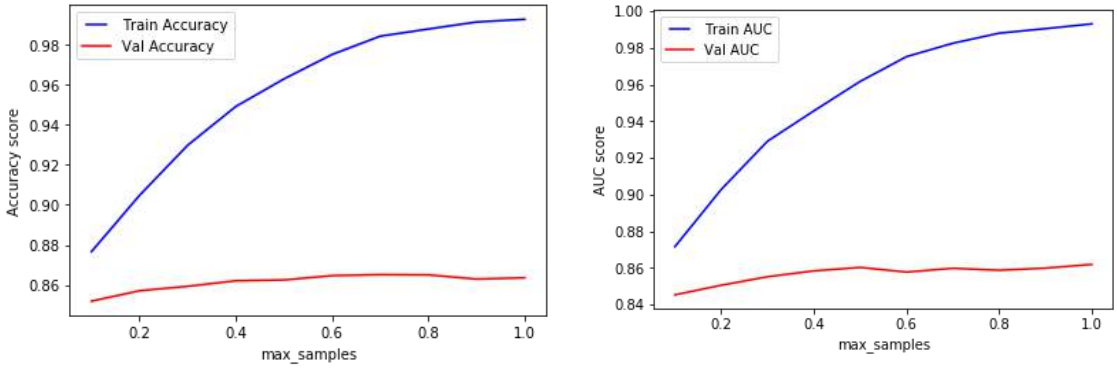


Ilustración 50 Ajuste de max_samples

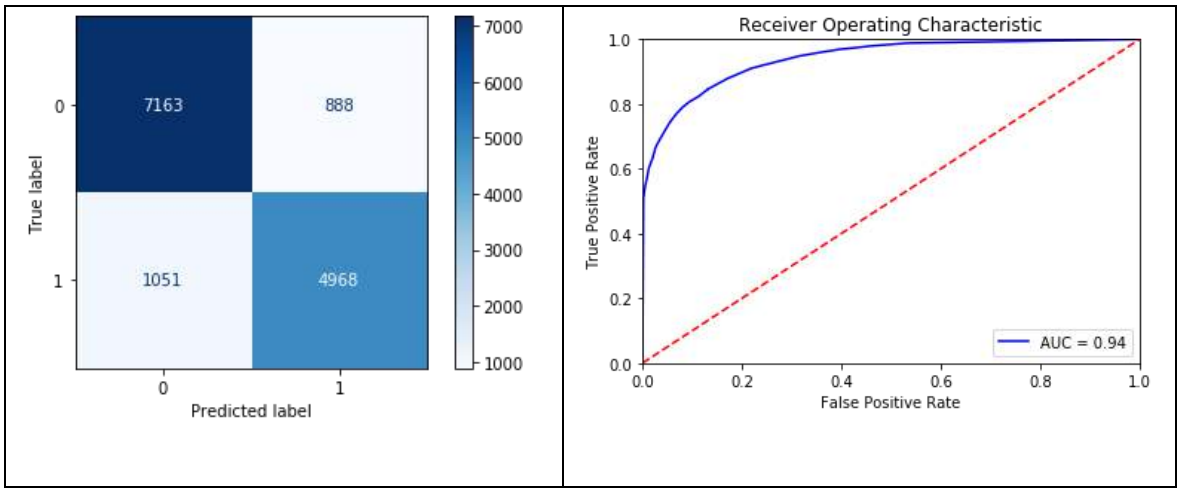
Tras observar los resultados, se ha decidido seleccionar el valor 1.0 para el max_samples. Este valor coincide con el valor que está definido por defecto, e indica que cada bolsa del bagging estará formada por el mismo número de datos que tiene el conjunto de entrenamiento. Hay que recordar que la bolsa de datos se crea con la técnica de boostrap con remplazamiento.

16.3 MODELO FINAL

Tras definir los hiperparámetros y los atributos que formaran el modelo, ya se puede entrenar el algoritmo. Antes de entrenar con los datos de entrenamiento se debe definir el modelo con sus hiperparámetros definidos:

```
BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=False,
                  max_features=1.0, max_samples=1.0, n_estimators=25,
                  n_jobs=None, oob_score=False, random_state=None, verbose=0,
                  warm_start=False)
```

Tras definir el modelo, se entrenan el algoritmo con los datos de entrenamiento. Una vez entrenado, se deben obtener los resultados de las prestaciones. Este paso se realiza con los datos de test, ya que estos datos no han sido visto nunca por el modelo, consiguiendo así, un resultado real de los resultados del modelo. Los resultados se muestran en la ilustración de 51 de la siguiente página:



Modelo	Accuracy	Bal. acc	Roc Auc	F1 Score	Sensitivity	Specificity
Bagging	0.86	0.85	0.94	0.84	0,82	0,89

Ilustración 51 Resultados finales bagging

17 MEJORANDO RANDOM FOREST

Al igual que en los otros dos apartados, en este apartado se trabajará con el algoritmo Random Forest para encontrar el mejor modelo posible para nuestro problema. También se realizará en dos partes: selección de los atributos más importantes y ajuste de los hiperparámetros propios del algoritmo. La separación de los datos ha sido la siguiente: 60% de entrenamiento, 20% de validación y finalmente 20% para el test.

17.1 SELECCIÓN DE ATRIBUTOS

Se ha realizado un estudio de los atributos más importantes. Para esto, se han creado 3 modelos, uno con 22 atributos, otro con 12 atributos y el ultimo con 8 atributos. Se han ido creando estos modelos de manera iterativa, eliminando cada vez los atributos que eran menos importantes. Para hacer la evaluación de la importancia en cada modelo se ha utilizado la técnica de cross validation, que nos permite tener unos resultados más reales y robustos.

Las importancias de los atributos que se han conseguido en cada modelo han sido:

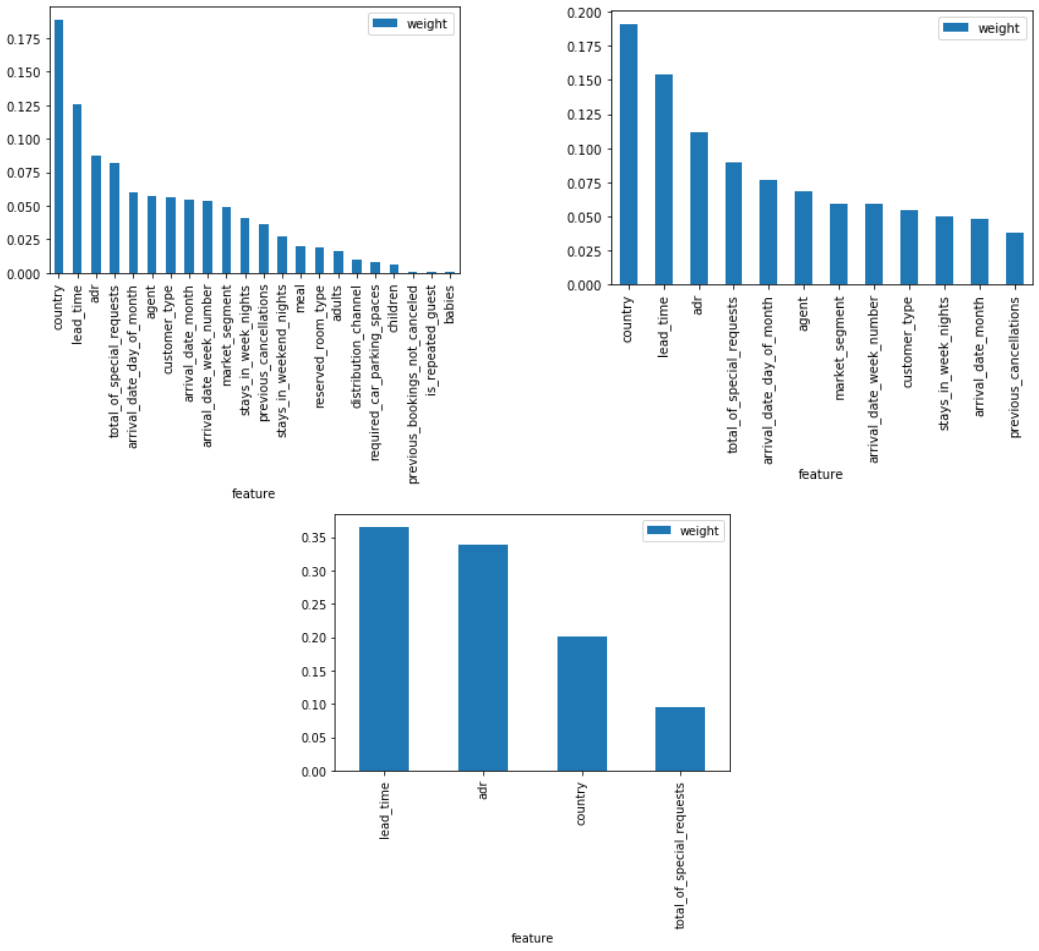


Ilustración 52 Importancia de atributos Random Forest

A continuación, en la ilustración 53, se compara la efectividad de cada uno de los modelos:

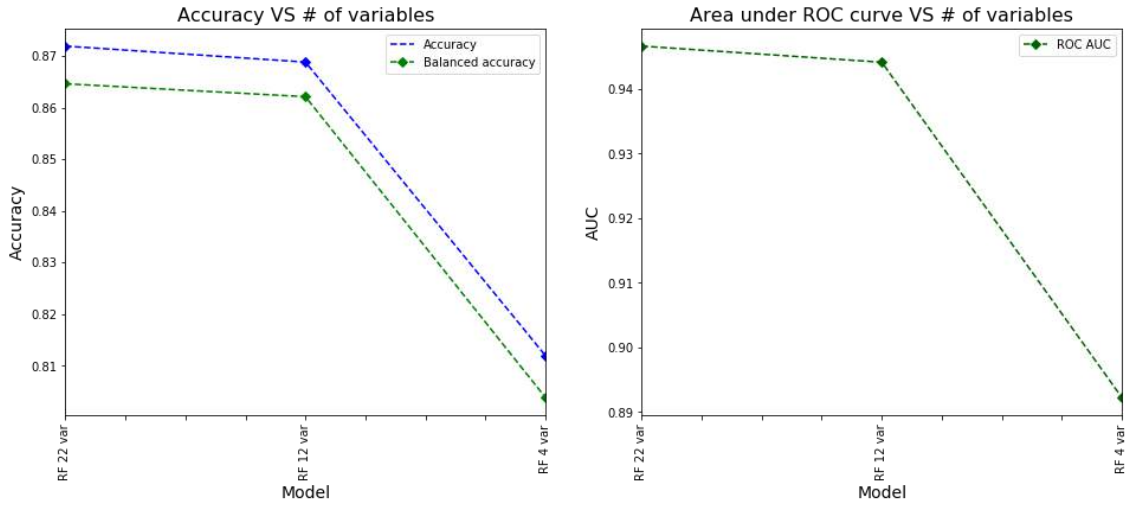


Ilustración 53 Accuracy y AUC por cada modelo

Con los resultados mostrados en el gráfico, se ha decidido utilizar el modelo de los 12 atributos, ya que las prestaciones de efectividad son muy buenas, y, además se reducen notablemente el tiempo de computación. Los 12 atributos que forman el modelo son:

- lead_time,
- arrival_date_week_number
- arrival_date_day_of_month
- stays_in_week_nights,
- previous_cancellations
- agent
- adr
- total_of_special_requests
- arrival_date_month
- country
- market_segment
- customer_type

Son los mismos atributos que forman tanto el modelo del árbol de decisión como modelo de bagging.

17.2 AJUSTE DE HIPERPARÁMETROS

En este apartado se ajustarán los parámetros más importantes del algoritmo de Random Forest. Para observar si las prestaciones del modelo mejoran o no, se utilizan los datos de validación. Los hiperparámetros que se ajustaran son:

- n_estimators
- max_depth
- max_features
- max_samples

n_estimators

El primer hiperparámetro que se ajustará es el n_estimators. Este parámetro hace referencia al número de bolsas de datos que queremos se creen, y, en consecuencia, el número de árboles de decisión. Por defecto, el valor definido es 100, a continuación, en la ilustración 54, se muestra cómo evoluciona la prestación del modelo Random Forest en función del n_estimators:

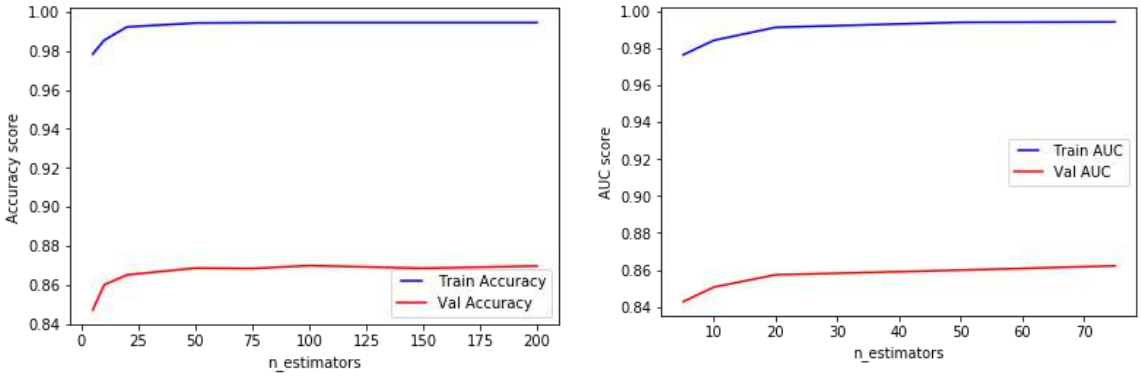


Ilustración 54 Ajuste de numero de estimadores

Se observa que el modelo está sobre ajustado, ya que se comportan demasiado bien con los datos de entrenamiento. Gracias a los datos de validación, se puede decir que un valor correcto para el numero de estimadores es 100, que coincide con el valor por defecto del modelo. Aumentado este valor no conseguimos un incremento en las prestaciones.

max_depth

Este parámetro hace referencia a la profundidad máxima que queremos que tenga cada uno de los árboles de decisión que se generan. A continuación, se detallan los resultados obtenidos:

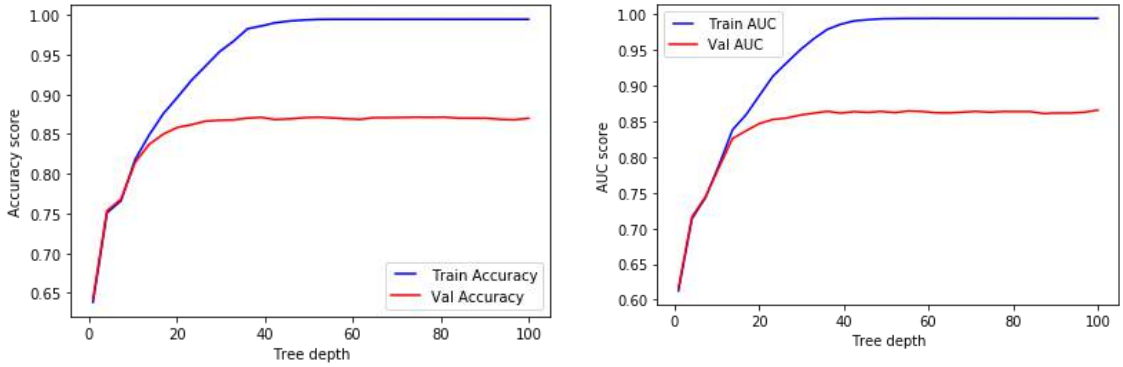


Ilustración 55 Ajuste max_depth

Tal y como se ve en la ilustración 55, se observa que el modelo pasa a tener un sobre ajuste cerca del valor 20, aunque las prestaciones de los datos de validación se mantengan similares. Ajustamos dicho parámetro a un valor de 100.

max_features

Este parámetro indica cuantos parámetros queremos que se usen para ser elegidos en cada uno de los árboles de decisión. Recordemos que en el Random Forest, también se llevaba a cabo una selección de variables. Se usa un valor entre 0 y 1 para mostrar la proporción de atributos que se usaran respecto a número total de atributos. Los resultados obtenidos son los de la ilustración 56:

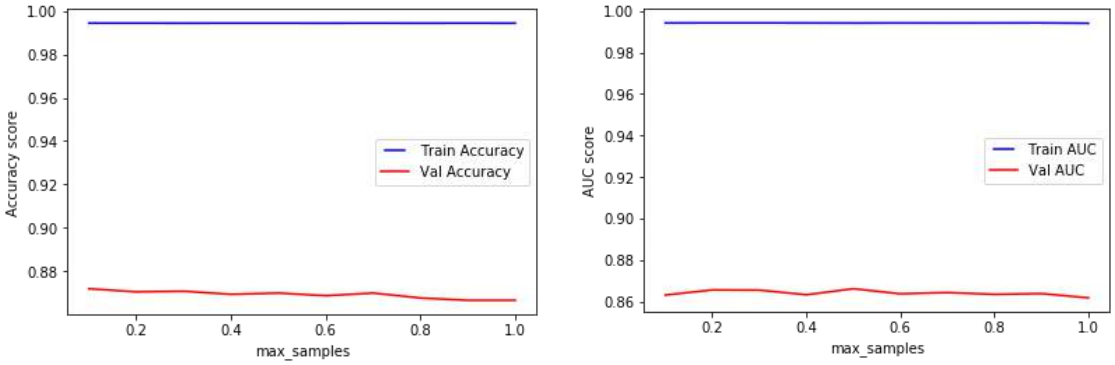


Ilustración 56 Ajuste max_features

Se decide quedarnos con el valor de max_features=0,2.

max_samples

Tras tener el n_estimators, max_depth y max_features definidos, se ha ajustado el parámetro max_samples. Este valor hace referencia a la cantidad que queremos por el cual este formado cada bolsa de datos del Random Forest. El valor es un valor del 0 al 1, y es la proporción de datos que formaran la bolsa en función de los datos totales de entrenamiento.

Se han obtenido los siguientes resultados de la ilustración 57:

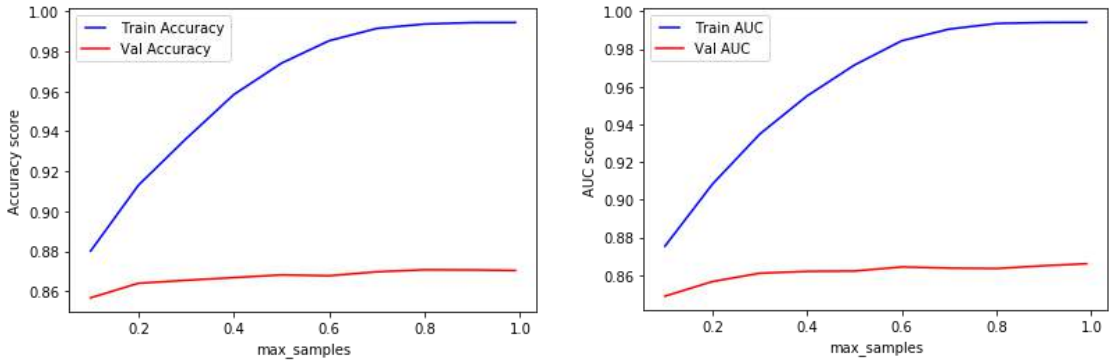


Ilustración 57 Ajuste max_samples

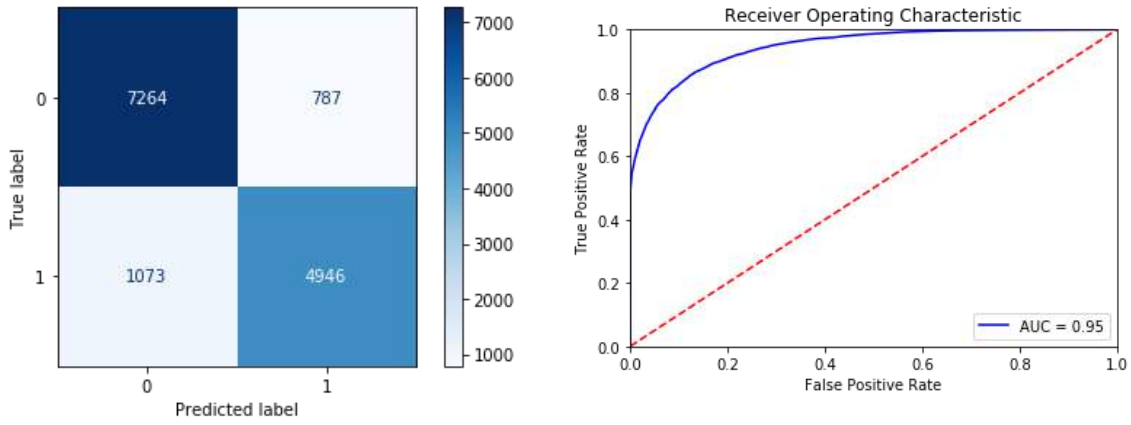
Se escoge el valor de max_samples=100. Con este valor se observa que el modelo aprende demasiado de los datos de entrenamiento. Pero, el modelo de validación no sufre ningún cambio y se mantiene igual o incluso mejor.

17.3 MODELO FINAL

Tras definir los hiperparámetros y los atributos que formaran el modelo, ya se puede entrenar el algoritmo. Antes de entrenar con los datos de entrenamiento se debe definir el modelo con sus hiperparámetros definidos:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=100, max_features=0.2,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Tras definir el modelo, se entrenan el algoritmo con los datos de entrenamiento. Una vez entrenando, se deben obtener los resultados de las prestaciones. Este paso se realiza con los datos de test, ya que estos datos no han sido visto nunca por el modelo, consiguiendo así, un resultado real de los resultados del modelo. Los resultados obtenidos se muestran en la ilustración 58 siguiente:



Modelo	Accuracy	Bal. acc	Roc Auc	F1 Score	Sensitivity	Specificity
Random Forest	0.867	0.862	0.946	0.842	0,82	0,90

Ilustración 58 Resultados finales Random Forest

Se podría llegar a crear un modelo que no tuviese sobreajuste con los datos de entrenamiento, pero se consigue unas prestaciones peores. Para conseguirlo se ajustarían max_samples=0.1, max_depth=20. A continuación, en la ilustración 59, se muestra cómo se comporta el modelo en función del n_estimators:

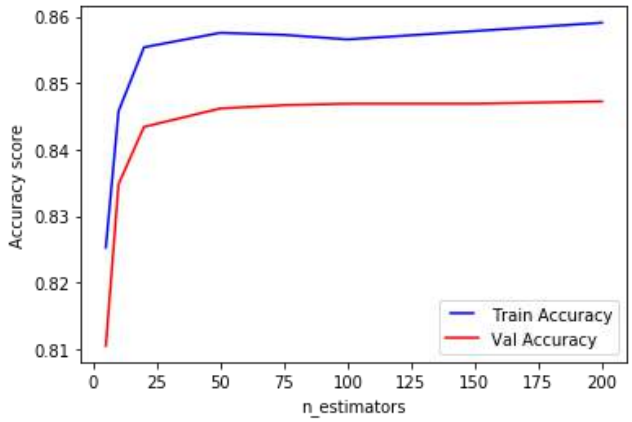


Ilustración 59 Ajuste de n_estimators con max_samples=0.1 y max_depth=20

Conseguimos evitar el sobre ajuste, pero obtenemos una peor efectividad: 0,845 contra el anterior de 0,867.

18 MEJORANDO ADABOOST

En este último apartado se estudiará en detalle la técnica de boosting con el algoritmo AdaBoost. Es importante recordar que este algoritmo está formado por múltiples arboles de longitud 1, por lo que el primer parámetro llamado base_estimator no será modificado porque es el valor que toma por defecto. En este apartado, al igual que los anteriores, se seleccionarán los mejores atributos para intentar reducir los tiempos de computación y finalmente se seleccionarán los hiperparámetros que se ajusten mejor a nuestro problema.

Los datos se han dividido en 60% de entrenamiento, 20% validación y 20% de test.

18.1 SELECCIÓN DE ATRIBUTOS

Como en los otros modelos, para seleccionar los mejores atributos, se ha usado la importancia de los atributos. Con esta información, se han generado un total de tres modelos y se han comparados sus prestaciones. El primer modelo constaba de 22 atributos, el segundo de 13 y el tercero de 4.En la ilustración 60, se muestran la importancia de cada uno de los atributos de cada modelo:

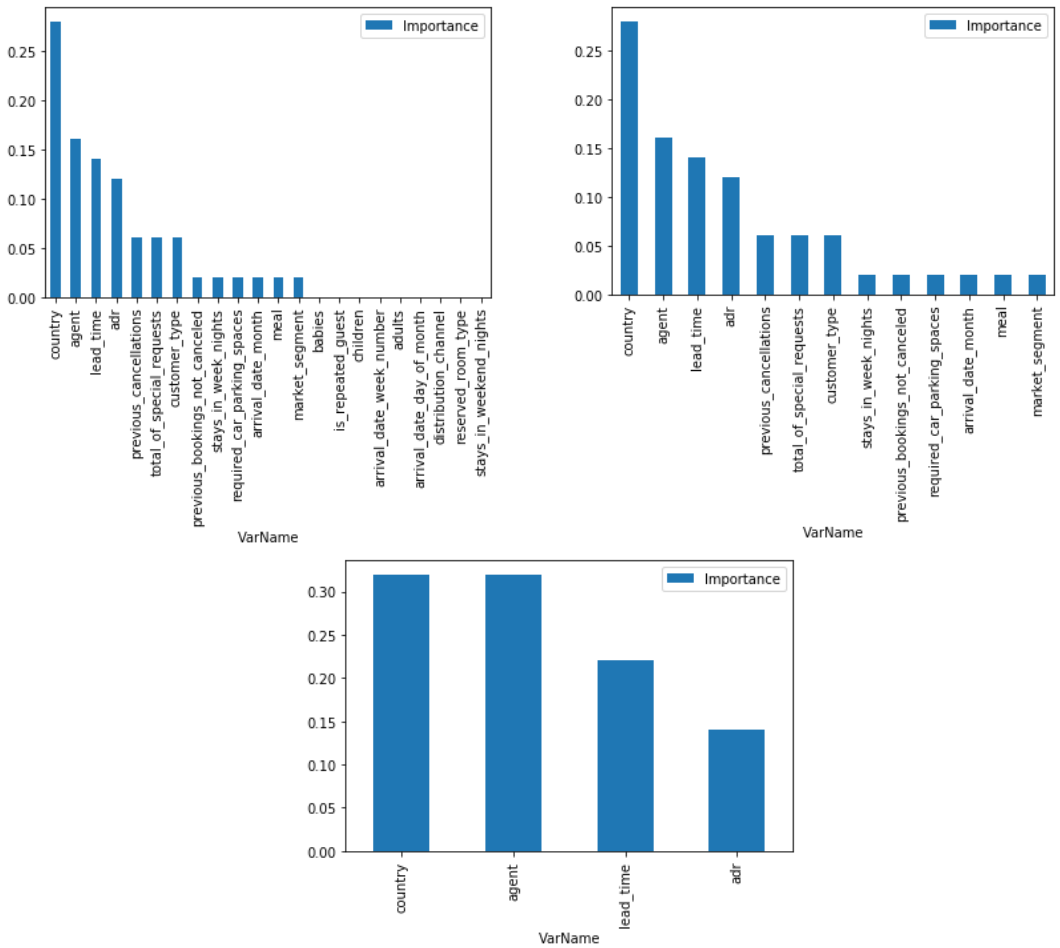


Ilustración 60 Importancia de atributos

Tras tener cada uno de los 3 modelos entrenados, se han comprobado sus prestaciones utilizando la técnica de 10-cross validation en el conjunto de datos de entrenamiento. Los resultados han sido los observados en la ilustración 61 siguiente:

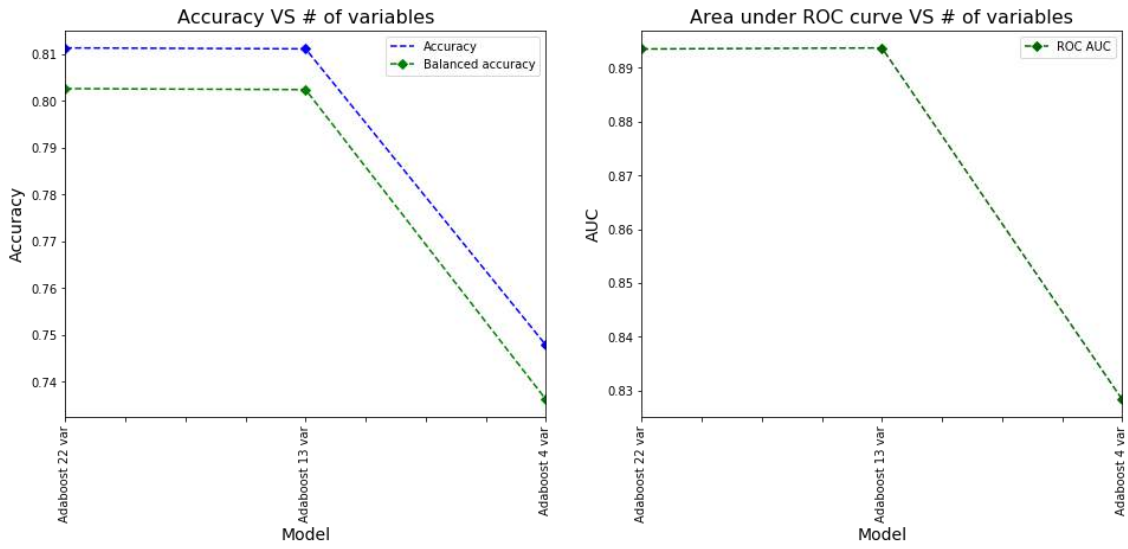


Ilustración 61 Accuracy y AUC de cada model

Visto las prestaciones similares del modelo de 22 atributos y el de 13, se ha decidido seguir trabajando con el modelo de 13 atributos, ya que se mantiene la efectividad y se reduce notablemente el tiempo de computación. Los 13 atributos que forman el modelo son:

- lead_time
- stays_in_week_nights
- previous_cancellations
- previous_bookings_not_canceled
- agent
- adr
- required_car_parking_spaces
- total_of_special_requests
- arrival_date_month
- meal
- country
- market_segment
- customer_type

18.2 SELECCIÓN DE HIPERPARÁMETROS

Tal y como se ha recordado en el apartado inicial de este sub apartado, los árboles de decisión que forman el algoritmo AdaBoost son de profundad máxima. Esta característica está definida así por defecto en el hiperparámetro base_estimator, por lo que este parámetro no se modificara. Se modificarán dos parámetros: n_estimators y learning_rate.

n_estimators

Este parámetro hace referencia al número de árboles de decisión que queremos que forme el algoritmo. El valor por defecto es 50. A continuación se muestra una gráfica en la ilustración 62 comparando las prestaciones del modelo en función del número de estimadores:

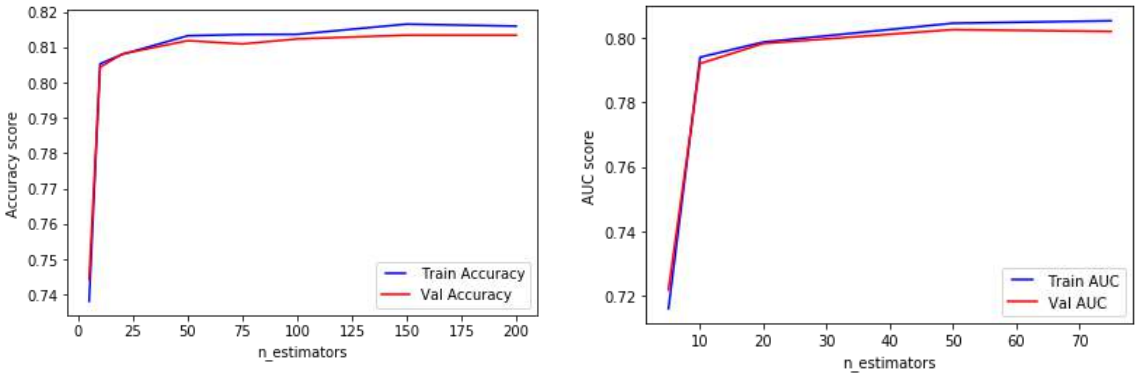


Ilustración 62 Ajuste de n_estimators

Se define el valor perfecto para n_estimators=100.

learning_rate

Este parámetro nos indica cuanto queremos que el modelo aprenda de cada uno de los árboles individuales creados. Un valor pequeño es mejor que un valor grande, pero un valor demasiado pequeño dificultar demasiado el entrenamiento. En la siguiente grafica se observan las prestaciones del modelo en función de learning_rate:

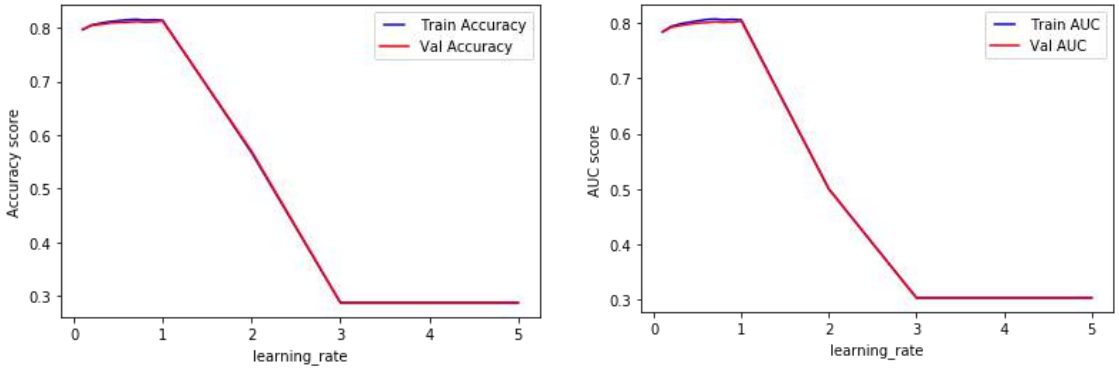


Ilustración 63 Ajuste learning_rate

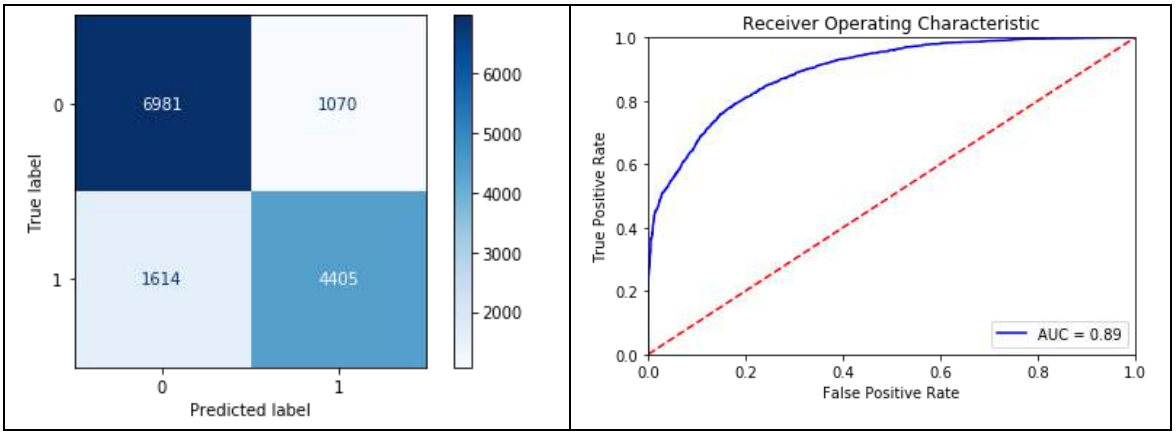
Se selecciona el valor de learning_rate=1. Este valor, es el que estaba asignado por defecto al modelo.

18.3 MODELO FINAL

Ahora ya se puede empezar a evaluar los resultados con los datos de test de nuestro modelo de 13 atributos. Pero antes se debe definir correctamente el modelo AdaBoost con los hiperparámetros que se han encontrado:

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1,
                   n_estimators=100, random_state=None)
```

Una vez ya definido el modelo, se pueden entrenar el modelo con los datos de entrenamiento y, finalmente, evaluar las prestaciones del modelo con los datos de test:



Modelo	Accuracy	Bal. acc	Roc Auc	F1 Score	Sensitivity	Specificity
AdaBoost	0.809	0.799	0.894	0.766	0.73	0.87

Ilustración 64 Resultados finales AdaBoost

19 COMPARACIÓN FINAL

Tras haber ajustado cada uno de los cuatros modelos diferentes, se comparan estos en función de los diversos parámetros que se han obtenido:

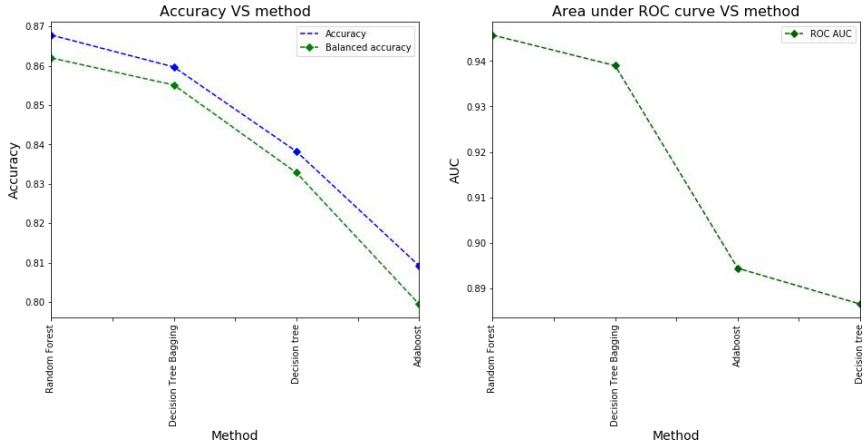


Ilustración 65 Accuracy y AUC de los modelos finales

En la ilustración 65, en el primer gráfico, se observa que, en todos los modelos, la exactitud es siempre un poco mayor a la exactitud balanceada. Esto se debe a que el modelo no está del todo balanceado. En cuanto a prestaciones de exactitud, se observa claramente que el mejor modelo de todos es el Random Forest. El árbol de decisión bagging parece ser el segundo mejor, con valores muy parecidos al Random Forest. En tercera posición se encuentra el árbol de decisión simple, con valores inferiores a los dos modelos anteriores comentados. Finalmente, y desde el punto de vista de exactitud, el modelo de boosting AdaBoost, es el que consigue peores prestaciones tanto en exactitud balanceada como en exactitud normal.

Con el segundo gráfico, se sigue la misma tendencia del primer gráfico, y es que el modelo Random Forest y el árbol de decisión bagging son los que mejores se comportan, es decir, los que tienen un área bajo la curva ROC mayor. A diferencia del primer gráfico, el algoritmo AdaBoost consigue unas prestaciones mejores que el árbol de decisión en cuanto al área de la curva ROC.

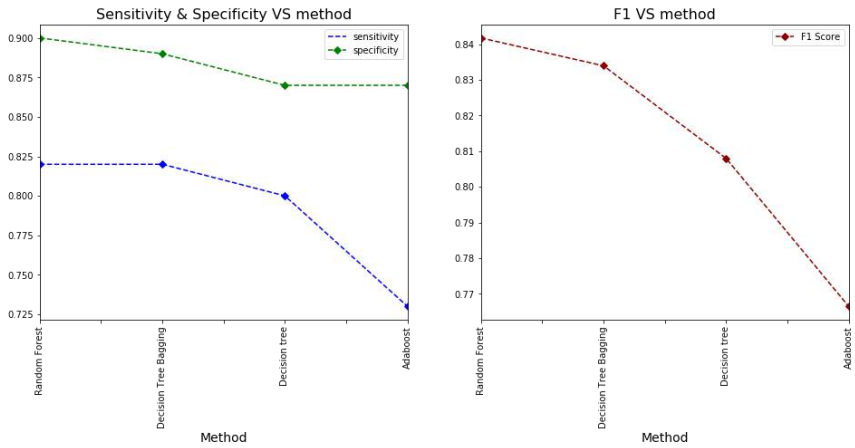


Ilustración 66 Sensitivity, Specificity y F1 por cada modelo final

En la ilustración superior 66, en el gráfico de la izquierda se puede observar como se comporta la sensibilidad y la especificidad de cada uno de los modelos. La sensibilidad hacer referencia a cómo se comporta el modelo prediciendo las cancelaciones y la especificidad hace referencia a cómo se comporta el modelo prediciendo las reservas no canceladas. Se puede observar que todos los modelos predicen mejor las reservas no canceladas que las canceladas. El Random Forest y el bagging tienen comportamientos muy parecidos, la única diferencia es que el bagging predice levemente peor las reservas no canceladas (especificidad). Se observa una gran diferencia con el AdaBoost, ya que este algoritmo tiene valores muy similares de especificidad con los otros modelos, pero, en cambio, tiene unas prestaciones muy deficientes en la sensibilidad, es decir, en acertar si una reserva fue cancelada o no.

El último grafico nos indica cómo se comporta cada modelo en términos del F1. Este parámetro relaciona la precisión y la sensibilidad, dicha magnitud es muy útil para entender el comportamiento de un modelo con datos nos balanceados. Se puedes observar, que los mejores modelos son Random Forest y Bagging. En un segundo escalón, se encontraría el árbol de decisión. Finalmente, el que obtiene un peor comportamiento es el modelo de boosting AdaBoost.

Finalmente, y tras analizar todos los gráficos y resultados, se puede decir que el mejor modelo para nuestro conjunto de datos es el Random Forest. Los otros modelos se ordenarían de la siguiente manera: bagging, árbol de decisión y AdaBoost. Siendo el modelo de Adaboost el que obtiene unas prestaciones más deficientes de los cuatro modelos. El resumen de las métricas de los modelos finales se muestra en la siguiente tabla5:

Tabla 5 Resumen de modelos finales

	Model	Accuracy	Balanced accuracy	ROC AUC	F1 Score	sensitivity	specificity
0	Decision tree	0.838237	0.832842	0.886554	0.807965	0.80	0.87
1	Decision Tree Bagging	0.859701	0.855119	0.938985	0.833922	0.82	0.89
2	Random Forest	0.867804	0.861990	0.945698	0.841729	0.82	0.90
3	Adaboost	0.809240	0.799473	0.894435	0.766487	0.73	0.87

Al iniciar la parte práctica, se había detectado que el hotel perdía una media de 250.000€ de ingresos en reservas canceladas los últimos 10 días. Con la aplicación del modelo Random Forest, se conseguiría detectar con éxito que reservas serán canceladas en un futuro, en concreto, predeciríamos bien el 87% de las reservas. Lo que se traduce en que las previsiones para los próximos años, solo se perderían ingresos valorados en aproximadamente el 30.000€.

20 CONCLUSIONES

Gracias a este trabajo se ha podido diferenciar cómo se comportan los principales algoritmos de aprendizaje supervisado en los problemas de clasificación utilizando, árboles de decisión.

Se han utilizado 4 modelos diferentes en un mismo conjunto de datos: árbol de decisión, bagging, Random Forest y AdaBoost. En cada uno de ellos, se ha detectado que el modelo con todos los atributos posibles y el mismo modelo con menos atributos tenían unas prestaciones muy similares. De esta manera, se consiguen modelos mucho menos pesados computacionalmente hablando y con unas prestaciones igual de buenas.

Se ha detectado la importancia de entender a la perfección el conjunto de datos que se tiene y cuál es el objetivo del problema. De esta manera se han podido eliminar algunos atributos y datos que nos hubiesen hecho un ajuste del modelo no real o práctico.

Al entrenar los modelos con Python, ha sido muy importante ajustar y entender cada uno de sus hiperparámetros. Cada tipo de algoritmo tiene sus parámetros diferentes y, en función del valor que se le asigne, el modelo puede presentar unos resultados muy dispares. Es por ese motivo, que la parte de selección de hiperparámetros ha sido una de las más importantes y críticas a la hora de encontrar el mejor modelo.

Cuando se realizan proyectos de aprendizaje automático y se quiere evaluar sus prestaciones, es indispensable diferenciar los datos de entrenamiento con los datos de test. En nuestro caso, en que se debían ajustar algunos parámetros, se ha añadido otro tipo de datos, los de validación. Gracias a estos se ha conseguido validar como se comportaba el modelo sin usar los datos de entrenamiento o test. De esta manera, se podía observar cómo variaban las prestaciones del modelo con nuevos datos, pero sin ajustar el modelo a los datos de test. Sin realizar este apartado, se podrían haber alcanzado problemas de sobre ajuste.

En el conjunto de datos trabajado, se ha observado unas mejores prestaciones en los algoritmos de métodos de conjuntos de bagging (Random Forest y Árbol de decisión Bagging). Por otro lado, el algoritmo de boosting AdaBoost ha obtenido unas prestaciones por debajo de los otros modelos. El modelo que ha conseguido unas mejores prestaciones ha sido el Random Forest, seguido muy de cerca por el bagging. El tercer mejor modelo ha sido el árbol de decisión simple y el ultimo ha sido el modelo AdaBoost. El modelo AdaBoost presentaba una buena especificidad, pero en cambio, una muy mala sensibilidad, por ese motivo, las prestaciones no han sido tan buenas.

En el caso práctico, se habían aproximado unas pérdidas de unos 250.000€ al año. Con la aplicación del modelo de Random Forest, se conseguiría reducir este valor hasta llegar a tener únicamente 35.000€ de pérdidas de ingresos anuales. La pérdida de ingresos se ha reducido alrededor del 87%. Por ese motivo, se puede considerar que tanto el modelo como el trabajo han sido un éxito.

El modelo generado solo predice las cancelaciones de las reservas del hotel de ciudad, pero se podría plantear un modelo capaz de predecir si una reserva será cancelada o no de múltiples hoteles. Se podría vender este modelo a grandes compañías hoteleras, como podrían ser por ejemplo NH o Hilton. La aplicación de dicho modelo ayudaría a estas compañías a mejorar las previsiones de las cancelaciones y a poder generar posibles estrategias para evitarlas.

A la hora de observar las prestaciones de un modelo, se debe mirar más de una métrica, ya que la "accuracy" puede ser un valor demasiado genérico y no preciso. Observando métricas como la curva ROC, la matriz de confusión o la puntuación F1 se ha podido entender mucho mejor como se comportaba cada uno de los modelos.

Finalmente, y para concluir, se puede decir que, para nuestro conjunto de datos, el modelo que mejor se ha comportado ha sido el Random Forest. Pero esto no significa que este sea el mejor modelo para cualquier conjunto de datos. Cada conjunto de datos se adaptará de manera distinta a cada uno de los modelos.

21 COSTE DEL PROYECTO

En este apartado se recoge una valoración de los costes de la realización de este trabajo. En este trabajo únicamente han aparecido costes de mano de obra (valorados de manera subjetiva en 22€/hora), ya que no ha habido ningún tipo de coste de material

Tabla 6 Coste del proyecto

	HORAS DEDICADAS (H)	PRECIO (€)
TEORIA	50	1100
PRACTICA	70	1540
REDACCIÓN	40	880
TOTAL	160 h	**3.520 €**

Tal y como se observar en la tabla 6 superior, se ha dividió las horas trabajas en: teoría, practica, y redacción del trabajo. La teoría hace referencia a las horas dedicadas a la búsqueda y afianzamiento de los conceptos de cada una de las técnicas, conceptos o modelos que se te tratan en el trabajo. Para ello se han usado artículos científicos y enlaces web.

El apartado de práctica hace referencia a todas aquellas horas que se han dedicado a la programación y realización práctica del proyecto. El tiempo es bastante elevado debido a los altos costes de computación que se requerían.

Finalmente, las horas de dedicación del apartado redacción hacen referencia al tiempo invertido para la redacción y perfeccionamiento del trabajo escrito.

Con estos tres tipos de costes, se concluye que se ha dedicado un total de 160 horas, lo que, con un coste de 22 €/hora, equivale a un coste económico total del proyecto de 3.520€.

22 BIBLIOGRAFÍA

22.1 PAPERS O ARTÍCULOS

S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," in IEEE Transactions on Systems, Man, and Cybernetics, vol. 21, no. 3, pp. 660-674, May-June 1991, doi: 10.1109/21.97458.

R. E. Banfield, L. O. Hall, K. W. Bowyer and W. P. Kegelmeyer, "A Comparison of Decision Tree Ensemble Creation Techniques," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, no. 1, pp. 173-180, Jan. 2007, doi: 10.1109/TPAMI.2007.250609.

L. Breiman, "Bagging Predictors", Machine Learning, vol. 24, pp. 123-140, 1996.

Y. Freund and R. Schapire, "Experiments with a New Boosting Algorithm", Proc. 13th Nat'l Conf. Machine Learning, pp. 148-156, 1996.

Ali, Jehad, et al. "Random forests and decision trees." International Journal of Computer Science Issues (IJCSI) 9.5 (2012): 272.

Hastie, Trevor, et al. "Multi-class adaboost." Statistics and its Interface 2.3 (2009): 349-360.

Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016.

Mussard, Stéphane, Françoise Seyte, and Michel Terraza. "Decomposition of Gini and the generalized entropy inequality measures." Economics Bulletin 4.7 (2003): 1-6.

Fushiki, Tadayoshi. "Estimation of prediction error by using K-fold cross-validation." Statistics and Computing 21.2 (2011): 137-146.

Visa, Sofia, et al. "Confusion Matrix-based Feature Selection." MAICS 710 (2011): 120-127.

Bradley, Andrew P. "The use of the area under the ROC curve in the evaluation of machine learning algorithms." Pattern recognition 30.7 (1997): 1145-1159.

22.2 ENLACES PÁGINAS WEB

Rocca, Joseph. "A Simple Introduction to Machine Learning". Medium, 2019, https://towardsdatascience.com/introduction-to-machine-learning-f41aabc55264.

Edwards, Gavin. "Machine Learning | An Introduction". Medium, 2018, https://towardsdatascience.com/machine-learning-an-introduction-23b84d51e6d0.

Fumo, David. "Types of Machine Learning Algorithms You Should Know". Medium, 2017, https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861.

Heidenreich, Hunter. "What Are the Types Of Machine Learning?". Medium, 2018, https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f.

Brownlee, Jason. "Supervised and Unsupervised Machine Learning Algorithms". Machine Learning Mastery, 2019, https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

"Árbol De Decisión En Machine Learning (Parte 1) - Sitiobigdata.Com". Sitiobigdata.Com, 2019, http://sitiobigdata.com/2019/12/14/arbol-de-decision-en-machine-learning-parte-1/#.

Nagpal, Anuja. "Decision Tree Ensembles- Bagging and Boosting". Medium, 2017, https://towardsdatascience.com/decision-tree-ensembles-bagging-and-boosting-266a8ba60fd9.

Demir, Necati. "Ensemble Methods: Elegant Techniques to Produce Improved Machine Learning Results". Toptal Engineering Blog, 2016, https://www.toptal.com/machine-learning/ensemble-methods-machine-learning.

"Boosting with Adaboost And Gradient Boosting". Medium, 2018, https://medium.com/diogo-menezes-borges/boosting-with-adaboost-and-gradient-boosting-9cbab2a1af81.

Navlani, Avinash. "Adaboost Classifier in Python". Datacamp Community, 2018, https://www.datacamp.com/community/tutorials/adaboost-classifier-python.

"Sklearn.Tree.Decisiontreeclassifier — Scikit-Learn 0.23.1 Documentation". Scikit-Learn.Org, 2020, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

"Sklearn.Ensemble.Baggingclassifier — Scikit-Learn 0.23.1 Documentation". Scikit-Learn.Org, 2020, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html.

"3.2.4.3.1. Sklearn.Ensemble.Randomforestclassifier — Scikit-Learn 0.23.1 Documentation". Scikit-Learn.Org, 2020, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

"Sklearn.Ensemble.Adaboostclassifier — Scikit-Learn 0.23.1 Documentation". Scikit-Learn.Org, 2020, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html.