Ensamble de modelos basados en árboles

Inteligencia Artificial (IS) 2018/19 – Propuesta de trabajo

Juan Galán Páez

1. Introducción y objetivo

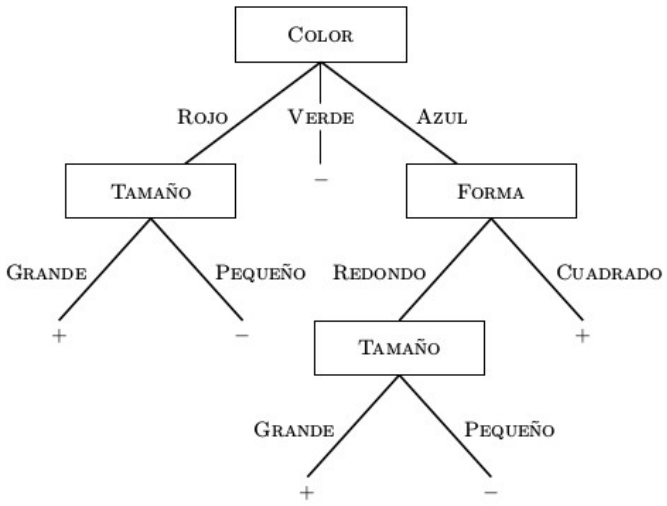
A continuación se introducen una serie de conceptos, que serán de utilidad para contextualizar el trabajo:

1.1. Algoritmos para la obtención de árboles de decisión

Los algoritmos para la obtención de árboles de decisión pertenecen al conjunto de técnicas de aprendizaje supervisado. Este tipo de algoritmos permiten obtener un árbol de decisión consistente con un conjunto de ejemplos de entrenamiento. A medida que se van añadiendo nodos al árbol, los ejemplos se van repartiendo entre las ramas (en función del valor del atributo que ha sido elegido como nodo) hasta llegar a las hojas, las cuales tendrán asignada una de las posibles categorías de la variable objetivo (también llamada variable respuesta).

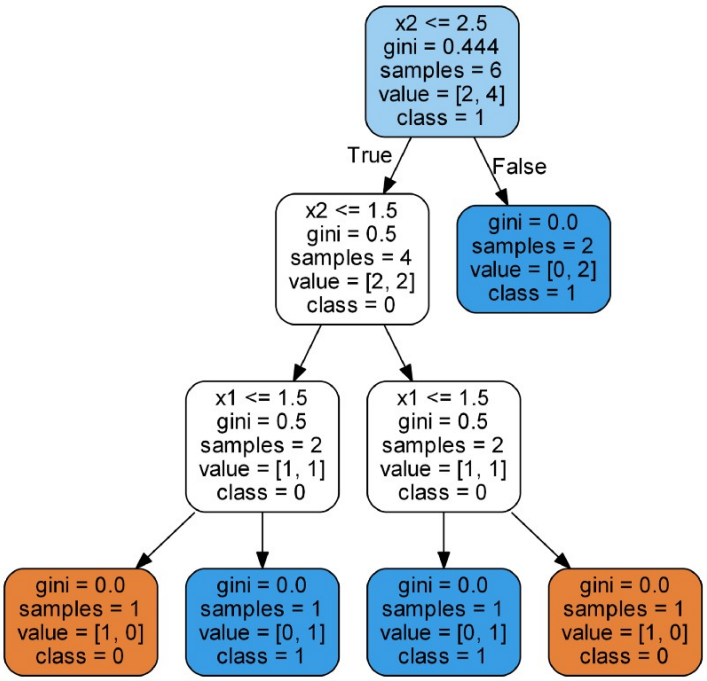
Para la realización de este trabajo usaremos un algoritmo que construye árboles binarios y cuya implementación nos proporciona la librería Scikit-learn de Python. Este algoritmo pertenece a la familia CART (Classification and Regression Trees). La principal diferencia con el algoritmo ID3 visto en clase, es que ID3 está pensado para trabajar con variables categóricas y CART permite trabajar tanto con variables continuas como categóricas.

El algoritmo ID3, permite expandir cada nodo en N ramas (donde N es el número de valores posibles que puede tomar la variable que representa a cada nodo). Esto implica que en nodos descendientes no volverá a usarse esta variable. Por ejemplo, en el árbol que se muestra a continuación, una vez que se ha usado el atributo *Tamaño* en la rama izquierda, este no se volverá a usar en sus descendientes. En el ejemplo vemos que el atributo *Tamaño* ha sido usado en un nivel inferior, pero en otra rama.



Al contrario que ID3, el algoritmo CART permite que sucesivos nodos descendientes vuelvan a usar

el mismo atributo. Esto se debe a que CART realiza cortes binarios (de cada nodo solo pueden generarse dos ramas), por lo que en cada nodo la variable no tiene por qué haber sido completamente explotada. Por ejemplo, en el árbol que se muestra a continuación vemos como la variable *x2* es usada en dos niveles sucesivos.



1.2. Sobreajuste y generalización

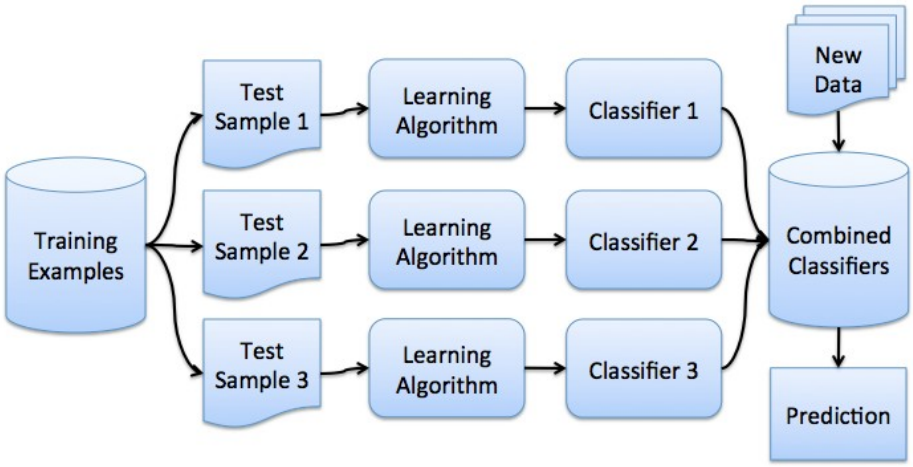
Los árboles de decisión son métodos muy flexibles y por tanto tienden a sobreajustar el conjunto de entrenamiento. Recordemos que cuando entrenamos un algoritmo sobre un conjunto de entrenamiento, nuestro objetivo es que el modelo obtenido sea válido en toda la realidad del problema, es decir, más allá de nuestros datos de entrenamiento. Decimos que un modelo tiene capacidad de generalización si es capaz de clasificar correctamente ejemplos que no pertenecen al conjunto de entrenamiento (aunque pertenecen al dominio del problema). Un modelo sobreajustado es aquel que solo clasifica correctamente los ejemplos del conjunto de entrenamiento.

La flexibilidad de los árboles de decisión permite que, si los dejamos crecer sin limitar su profundidad, pueden acabar aprendiendo cada instancia del conjunto de entrenamiento, en lugar de patrones generales (que es lo que buscamos). En un caso extremo de sobreajuste, el árbol aprendido tendría una hoja por cada ejemplo del conjunto de entrenamiento.

Existen numerosas técnicas para enfrentarnos al sobreajuste, como limitar la profundidad máxima del árbol o la poda a posteriori vista en clase. Sin embargo al limitar el sobreajuste, tenemos el riesgo de limitar la capacidad predictiva de nuestro modelo. El encontrar el equilibrio entre un modelo excesivamente sobreajustado y uno excesivamente general es uno de los grandes problemas del aprendizaje automático. A esto se le denomina el problema del sesgo y la varianza [1]. En este trabajo nos centramos en los métodos de ensamble [2] que tienen como objetivo obtener un modelo con buena capacidad predictiva sin caer en el sobreajuste, en la medida de lo posible.

1.3. Ensamble de algoritmos

La idea del ensamble de modelos consiste en entrenar varios modelos sobre un mismo problema y luego combinarlos para obtener un modelo final con mayor capacidad predictiva y menor sobreajuste que los modelos que lo componen. Existen numerosas formas (técnicas de ensamble) de agregar estos modelos y obtener una predicción final y más robusta. Las más sencillas consisten en calcular la predicción media o la más frecuente entre las predicciones de cada uno de los modelos.



Uno de los requisitos necesarios para obtener un buen ensamble de modelos es que estos sean diferentes entre si. En lineas generales existen dos formas de obtener modelos diferentes: o bien entrenando diferentes algoritmos (redes neuronales, regresión lineal, Knn, árboles de decisión, etc.) sobre el mismo conjunto de entrenamiento, o bien entrenando el mismo algoritmo sobre diferentes conjuntos de entrenamiento. En este trabajo nos centraremos en la segunda opción.

La idea detrás de esto es que, en vez de obtener un modelo que intente capturar correctamente toda la casuística del problema, obtenemos varios modelos, cada uno especializado en una parte del problema, que al ponerlos en común nos proporciona una solución más robusta. Esta idea es posible observarla en la realidad.

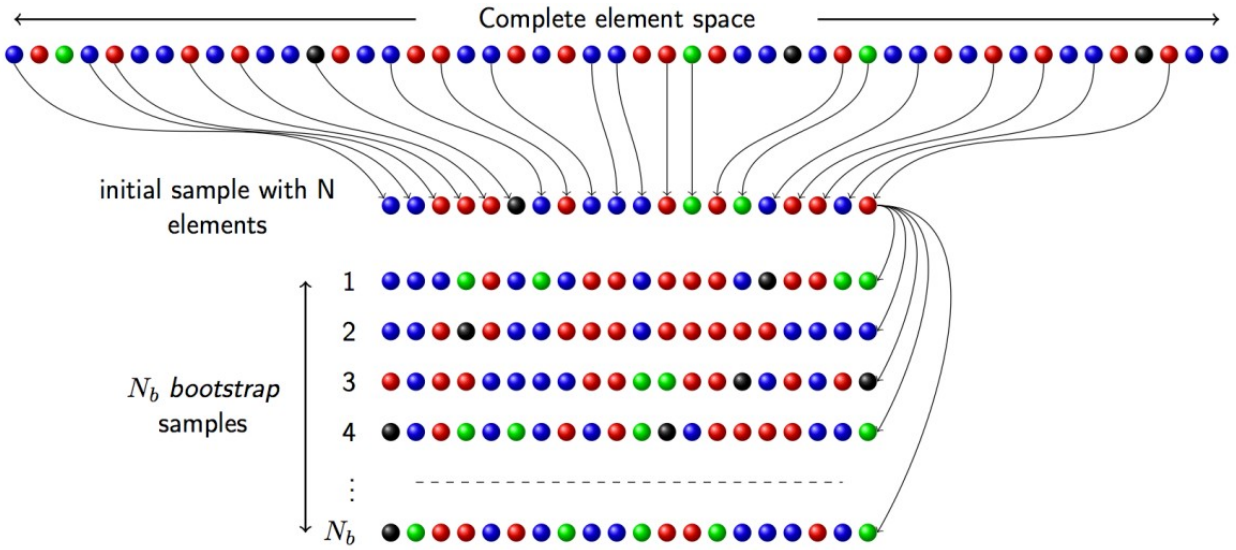
Supongamos que 10 analistas de bolsa intentan predecir si la acción de Telefónica va a subir o va a bajar. Para esto investigan en multitud de noticias e informes, los cuales es muy frecuente que contengan, información contradictoria, errónea o falsa. Además, ninguno de los analistas será capaz de procesar toda la información existente sobre Telefónica. El resultado es que no todos los analistas tienen por qué alcanzar la misma conclusión y por tanto, algunos alcanzarán conclusiones erróneas. La solución es no fiarse de un solo individuo, sino hacer una votación entre todos los analistas. Aunque alguno se equivoque, es más probable que el voto mayoritario sea el correcto.

Se ha comentado que una de las opciones existentes para obtener modelos diferentes entre si, consiste en entrenar nuestro algoritmo con diferentes conjuntos de datos. Esto plantea un problema y es que normalmente la cantidad de ejemplos que contiene nuestro conjunto de entrenamiento es limitada. Por tanto dividir el conjunto de entrenamiento en muchos subconjuntos y entrenar un algoritmo con cada subconjunto no es una opción. A continuación introducimos dos técnicas que nos ayudaran a resolver este problema.

1.4. Técnicas de muestreo para ensamble de modelos

Una de las técnicas más populares para derivar nuevos conjuntos de ejemplos a partir de la muestra

original se denomina **Bootstrapping** [3]. Esta es una técnica muy usada en estadística con el objetivo de conocer con fiabilidad la distribución de una población completa a partir de una muestra.
Boostrapping consiste en, a partir de un conjunto con N ejemplos (filas), obtener muchas muestras de tamaño N realizando muestreo con reemplazamiento sobre N.



Es decir, a partir de un conjunto de entrenamiento con N ejemplos, creamos muchos conjuntos diferentes (de tamaño N) tras realizar muestreo con reemplazamiento sobre el original. Estos conjuntos obtenidos, serán parecidos pero no idénticas ya que, como resultado del muestreo con reemplazamiento, algunos ejemplos del conjunto original aparecerán repetidos y otros no estarán presentes. Esto nos permitirá entrenar modelos diferentes usando siempre el total de la población de ejemplos.

La segunda técnica que usaremos en este trabajo se denomina **Random Subspace method**. Esta técnica es otra idea sencilla pero igualmente potente. Si en el caso anterior trabajamos a nivel de ejemplos (filas), en este caso lo haremos con las características o variables (columnas) de nuestro conjunto de entrenamiento. Recordemos nuestro objetivo, obtener modelos que hayan aprendido cada uno diferentes patrones, sobre el problema, para luego agregarlos y obtener un modelo. Para esto, el método random subspace consiste en entrenar cada uno de los modelos usando un subconjunto aleatorio de variables o características.

De esta forma, cada modelo deberá aprender lo máximo posible de solo una parte de la información. Algunos modelos habrán aprendido patrones usando unas variables del conjunto de datos y otro modelo otros patrones con otras variables distintas. O visto de otro modo, unos modelos serán especialistas en unas dimensiones del problema y otros en otras. De la combinación de estos modelos especializados se espera obtener un modelo más robusto.

Por último, merece la pena destacar una técnica de aprendizaje automático denominada Random Forest ya que es una de las más utilizadas y que tienen mayor capacidad predictiva un muchos tipos de problemas. Random Forest (bosques aleatorios) consiste en entrenar múltiples árboles de decisión, usando tanto bootstraping como el método random subspace (de una forma diferente a como lo haremos en este trabajo), para luego promediar las predicciones de cada uno de estos.

1.5. Objetivo principal

El **objetivo principal** de esta propuesta es desarrollar nuestro propio ensamble de árboles de decisión, es decir, una versión simplificada de Random Forests. Según lo visto anteriormente, el objetivo es aplicar las técnicas Bootstrapping y Random Subspace Method para obtener múltiples conjuntos de de entrenamiento sobre los que los que entrenar árboles de decisión para luego promediar sus predicciones. A partir de ahora llamaremos a nuestro ensamble de árboles, el **meta-algoritmo**. Este objetivo se descompone en los siguientes objetivos específicos:

- Todo el desarrollo realizado debe ser generalizable, es decir, no debe estar vinculado a un conjunto de datos o problema concreto. En concreto debe ser capaz de abordar diferentes problemas de clasificación binaria (la variable respuesta solo toma dos valores). Para esto, el código debe estar parametrizado según sea necesario (conjunto de entrenamiento, conjunto de evaluación, nombre de la columna que lleva la variable respuesta, etc.).
- Implementar un generador de conjuntos de datos, que a partir de un conjunto de datos de entrada, devuelva un conjunto de datos similar, según las técnicas descritas en la introducción.
- Implementar una función de entrenamiento para el meta-algoritmo. Esta función recibirá el conjunto de datos de entrenamiento y el número de árboles de decisión que se desean entrenar, además de otros parámetros de entrada que se especificarán en en la siguiente sección. Por cada árbol a entrenar se generará un conjunto de datos (usando el generador anterior) y se usará un algoritmo de entrenamiento sobre dicho conjunto de datos. No es necesario implementar el algoritmo de entrenamiento de árboles de decisión, para lo que se recomienda usar DecisionTreeClassifier[1] de Scikit-learn. Esta función devolverá una lista de árboles entrenados.
- Implementar una función de predicción para el meta-algoritmo. Esta función recibirá el conjunto de árboles entrenados (obtenido mediante la función de entrenamiento del meta-modelo) y un conjunto de evaluación. Obtendrá la predicción promedio de cada uno de los árboles sobre el conjunto test.
- Métricas de evaluación. Realizar la evaluación de las predicciones obtenidas sobre el conjunto de evaluación mediante el meta-algoritmo. Es frecuente que los problemas de clasificación binaria estén desbalanceados, es decir, en el conjunto de datos no existe el mismo número de ejemplos de cada una de las clases. Por esto, el rendimiento de nuestro algoritmo (capacidad predictiva sobre el conjunto de evaluación) medido como tasa de aciertos (aciertos/ejemplos totales) no es la más adecuada. Scikit learn proporciona, además de la tasa de aciertos (accuracy_score[2]), la tasa de aciertos balanceada que viene a solucionar el problema anteriormente mencionado (balanced_accuracy_score[3]).
- Se proporciona un conjunto de datos en el que se ha comprobado que es posible obtener una solución que mejore ligeramente el rendimiento del árbol de decisión simple. Se pide que se exploren diferentes parámetros de entrada del meta-algoritmo y se documenten aquellos que proporcionen el mejor rendimiento (métricas de rendimiento) sobre el conjunto de evaluación. Deberán usarse tanto la tasa de aciertos tradicional como la balanceada.
- Documentar el trabajo en un fichero con formato de artículo científico, explicando con precisión las decisiones de diseño en la implementación del meta-algoritmo. También se explorarán y documentarán las situaciones (parámetros de entrada) en las que nuestro meta-algoritmo tiene un buen desempeño y aquellas en las que el rendimiento sea malo.
- Realizar una presentación de los resultados obtenidos en la defensa del trabajo.

1 https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
2 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score
3 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html#sklearn.metrics.balanced_accuracy_score

Para que el trabajo pueda ser evaluado, se deben satisfacer TODOS los objetivos específicos al completo: el trabajo debe ser original, estar correctamente implementado y funcionar perfectamente, los experimentos se deben haber llevado a cabo y analizados razonadamente, el documento debe ser completo y contener entre 8 y 12 páginas (no más, no menos), y se debe realizar la defensa con una presentación de los resultados obtenidos.

2. Descripción del trabajo

A continuación se introduce la metodología a seguir para el correcto desarrollo del trabajo.

2.1. Implementación de los algoritmos

El uso de clases para construir el meta-algoritmo facilitará su diseño y usabilidad sin embargo, esto no es obligatorio.
Se pueden usar funciones ya implementadas para el cálculo de métricas o la obtención de muestras aleatorias a partir de conjuntos. No se permite el uso de funciones de alto nivel que realicen el entrenamiento del meta-algoritmo, la obtención de predicciones a partir del meta-modelo ni la obtención del nuevo conjunto de datos (tras aplicar bootstraping y random subspace method).

El algoritmo de entrenamiento tendrá como entrada, al menos los siguientes parámetros:

- Conjunto de entrenamiento: es el conjunto de entrenamiento original. Supongamos que tiene N filas y M columnas (excluyendo la variable objetivo del aprendizaje).
- Número de árboles: Tomará cualquier valor a partir de uno.
- Proporción de columnas: Es un número en el rango [0, 1] que indica la proporción de columnas (excluyendo la variable respuesta) o variables predictoras a considerar cuando se entrene cada árbol. Es decir, si tenemos 20 variables predictoras y fijamos una proporción de 0.75, cada árbol se entrenará usando 15 variables.

El parámetro *Número de árboles* indica el número de árboles de decisión a entrenar. Por cada árbol, el algoritmo de entrenamiento:

1. Generará un nuevo conjunto de datos con N filas mediante muestreo con reemplazamiento sobre las filas del *conjunto de entrenamiento*. Además, las columnas de este conjunto de datos serán un subconjunto (sin reemplazamiento) aleatorio de las M columnas del conjunto original.
2. Entrenará un árbol de decisión sobre ese conjunto de datos.

El resultado será el conjunto de árboles entrenados.

La función de predicción recibirá un conjunto de datos de evaluación (no usado en el entrenamiento) y proporcionará una predicción. Para esto, realizará la predicción de cada arbol entrenado en la fase anterior y luego obtendrá una predicción promedio o mayoritaria. Además, si se proporciona un conjunto de evaluación para el que se conoce la variable respuesta, se calcularán y proporcionarán las métricas:

- Tasa de aciertos.
- Tasa balanceada de aciertos.

Será necesario manipular tablas de datos (DataFrames), especialmente en la generación de conjuntos de datos. Para esto, se recomienda el uso de la librería Pandas de Python.
Las clases o funciones desarrolladas deben proporcionarse en un módulo python (.py) de forma que

puedan importarse y ser usadas en un notebook.

2.2. Experimentación

Para la fase de experimentación es recomendable la presentación de un notebook de Jupyter que facilite la realización e interpretación de los diferentes experimentos.
Para esta fase se proporciona el conjunto de datos *Adult Data Set*[4].Debe usarse el conjunto de datos que se proporciona junto con este documento y no el que está disponible en la web ya que se han realizado tareas de limpieza de datos e imputación de valores faltantes sobre el conjunto original.

El conjunto contiene información censal sobre individuos de Estados Unidos. El objetivo es, a partir de esta información, predecir si la persona gana más o menos de 50K$ al año. Las variables que contiene el conjunto de datos proporcionado son las siguientes (en orden de aparición):
["age", "capital-gain", "capital-loss", "hours-per-week", "workClass", "education", "marital-status", "occupation", "relationship", "race", "sex", "native-country", "income"]
Para más información sobre el significado de cada variable, acudir a la página del conjunto de datos. La variable objetivo es la última, "income".

Se proporcionan dos archivos, "adult_train.csv" y "adult_test.csv". El primer fichero es el de entrenamiento y el segundo el de evaluación. Esto significa, que el segundo fichero debe usarse únicamente para evaluar el rendimiento de los modelos. Y el primero para entrenarlos.

Como se ha comentado, los conjuntos de datos casi no necesitan preprocesado. Solo deben convertirse las variables categóricas (texto) a numéricas, tal y como se vio en la práctica 1. Dado que estamos usando métodos basados en árboles, no es necesario aplicar la codificación one-hot.

En la fase de experimentación, para tener un punto de referencia, debemos entrenar y evaluar un árbol de decisión simple. A partir de ahí, se explorarán diferentes parámetros de entrada para nuestro meta-algoritmo con el fin de obtener aquellos que mejor rendimiento proporcionen. Los parámetros a explorar son:

- Número de árboles
- Proporción de columnas
- Profundidad de los árboles (**opcional**): En el algoritmo DecisionTreeClassifier podemos limitar la profundidad máxima del árbol.
- Otros parámetros del algoritmo DecisionTreeClassifier (**opcional**): Aunque la profundidad es uno de los parámetros más importantes, existen otros como el número mínimo de ejemplos por hoja o el numero mínimo de ejemplos por nodo que también ayudan a limitar la profundidad del árbol.

Para obtener una estimación de rendimiento fiable (eliminando la variabilidad introducida por el muestro aleatorio) se recomienda medir el rendimiento (repetir el experimento) para cada combinación de parámetros mas de una vez y luego calcular el promedio.
Por último, si además de medir el rendimiento sobre el conjunto de evaluación, lo hacemos también sobre el conjunto de entrenamiento, podremos observar como el sobreajuste afecta de diferente manera a los modelos sencillos que al ensamble de modelos.

4 http://archive.ics.uci.edu/ml/datasets/adult

2.3. Documentación

En el fichero `plantilla-trabajo.doc` se muestra una sugerencia de estructura y formato de estilo artículo científico correspondiente a la documentación del trabajo. Este formato es el del *IEEE conference proceedings*, cuyo sitio web *guía para autores* [6] ofrece información más detallada y plantillas para Word y Latex.

El artículo deberá tener una extensión **entre 8 y 12 páginas**, y la estructura general del documento debe ser como sigue: en primer lugar realizar una **introducción** al trabajo explicando el objetivo fundamental, incluyendo un breve repaso de antecedentes en relación con la temática del trabajo. A continuación, describir la **estructura** del trabajo, las **decisiones de diseño** que se hayan tomado a lo largo de la elaboración del mismo, y la **metodología** seguida al implementarlo (nunca poner código, pero sí pseudocódigo), y seguidamente detallar los **experimentos** llevados a cabo, **analizando los resultados** obtenidos en cada uno de ellos. Por último, el documento debe incluir una sección de **conclusiones**, y una **bibliografía** donde aparezcan no sólo las referencias citadas en la sección de introducción, sino cualquier documento consultado durante la realización del trabajo (incluidas las referencias web a páginas o repositorios).

2.4. Mejoras

Aunque no sea obligatorio, sí se tendrán en cuenta en la calificación los siguientes objetivos adicionales:

- Los elementos marcados como opcional a lo largo de este documento.
- Adaptar la funcionalidad desarrollada para que se puedan entrenar y evaluar modelos, correctamente, sobre conjuntos de datos con variable respuesta multi-clase.
- Explorar otros conjuntos de datos y evaluar como se comporta el algoritmo desarrollado. Téngase en cuenta que en los conjuntos de datos muy sencillos (como el conjunto 'cars' visto en prácticas), los árboles de decisión no sufren problemas de sobreajuste y el rendimiento del algoritmo implementado sería inferior. Se recomienda usar conjuntos datos con almenos varias decenas de miles de filas y a partir de 15 o 20 variables predictoras.
- Nuestro algoritmo entrenará múltiples árboles al mismo tiempo. Estas tareas son totalmente independientes y se beneficiarían de una implementación en paralelo.

2.5. Presentación y defensa

El día de la defensa se deberá realizar una pequeña presentación (PDF, PowerPoint o similar) de 10 minutos en la que participarán activamente todos los miembros del grupo que ha desarrollado el trabajo. Esta presentación seguirá a grandes rasgos la misma estructura que el documento, pero se deberá hacer especial mención a los resultados obtenidos y al análisis crítico de los mismos. Se podrá usar un portátil (personal del alumno), diapositivas y/o pizarra. En los siguientes 10 minutos de la defensa, el profesor procederá a realizar preguntas sobre el trabajo, que podrán ser tanto del documento como del código fuente. Adicionalmente, el profesor podrá proporcionar un nuevo conjunto de datos con el que probar el algoritmo implementado.

3. Criterios de evaluación

Para que el trabajo pueda ser evaluado, se deberá satisfacer los objetivos concretos descritos en el apartado 1 (todos y cada uno de ellos, si no, el trabajo obtendrá una nota de suspenso). Uno de los alumnos del equipo deberá subir a través del formulario disponible en la página de la asignatura un fichero comprimido .zip, que contenga:

- **Una carpeta con el código fuente**. Dentro de dicha carpeta tiene que haber un fichero README.txt, que resuma la estructura del código fuente. Además se proporcionarán instrucciones sobre como reproducir la experimentación realizada. El código fuente puede ir acompañado de un notebook de Jupyter que contenga la parte de experimentación. Es importante la coherencia de este fichero con la defensa.
- **El documento – artículo en formato PDF**. Deberá tener una extensión mínima de 8 páginas, y máxima de 12. Deberá incluir toda la bibliografía consultada (libros, artículos, technical reports, páginas web, códigos fuente, diapositivas, etc.) en el apartado de referencias, y mencionarlas a lo largo del documento.

Para la evaluación se tendrá en cuenta el siguiente criterio de valoración, considerando una nota máxima de 3 en total para el trabajo:

- **El código fuente (1.5 punto) y experimentación**: se valorará la claridad y buen estilo de programación, corrección, eficiencia y usabilidad de la implementación, y calidad de los comentarios. La claridad del fichero README.txt también se valorará. En ningún caso se evaluará un trabajo con código copiado directamente de Internet o de otros compañeros.
  En este trabajo está permitido utilizar en parte librerías de Python existentes, aunque en este apartado se valorará exclusivamente el código original desarrollado por los alumnos.
  Con respecto a la experimentación, se valorará la calidad y completitud de los experimentos realizados. Además se tendrá en cuenta el rendimiento del algoritmo y el conjunto de mejores parámetros proporcionado. Por último, no se tendrán en cuenta aquellos resultados experimentales que no sean reproducibles.
- **El documento – artículo científico (1 punto)**:
  - Se valorará el uso adecuado del lenguaje y el estilo general del documento (por ejemplo, el uso de la plantilla sugerida).
  - Se valorará en general la claridad de las explicaciones, el razonamiento de las decisiones, y especialmente el análisis y presentación de resultados en las secciones de experimentación y conclusiones.
  - Igualmente, no se evaluará el trabajo si se detecta cualquier copia del contenido del documento. La sección de referencias deberá incluir menciones a todas las pertinentes fuentes consultadas.
- **La presentación y defensa (0,5 puntos)**: se valorará la claridad de la presentación y la buena explicación de los contenidos del trabajo, así como, especialmente, las respuestas a las preguntas realizadas por el profesor.
- **Mejoras**: Se valorarán hasta con 0.5 puntos extra sin superar el máximo de 3 puntos totales del trabajo.

IMPORTANTE: Cualquier **plagio**, **compartición de código** o uso de material que no sea original y del que no se cite convenientemente la fuente, significará automáticamente la **calificación de cero** en la asignatura para **todos los alumnos involucrados**. Por tanto, a estos alumnos **no se les conserva**, ni para la actual ni para futuras convocatorias, **ninguna nota** que hubiesen obtenido hasta el momento. Todo ello sin perjuicio de las correspondientes **medidas disciplinarias** que se pudieran tomar.

4. Referencias

[1] Equilibrio sesgo varianza. https://koldopina.com/equilibrio-varianza-sesgo/
[2] Métodos de ensamble. http://www.cs.us.es/~fsancho/?e=106
[3] Bootstrapping. https://es.wikipedia.org/wiki/Bootstrapping_(estadística)

[4] Random subspace method. https://en.wikipedia.org/wiki/Random_subspace_method
[5] Random Forests. https://es.wikipedia.org/wiki/Random_forest
[6] Plantilla de documento científico IEEE.