UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Estudio y Desarrollo de una Librería en R para Evaluar las Prestaciones de un Clasificador

TRABAJO FINAL DE MÁSTER

Máster Universitario en Gestión de la Información

Autor: Paulina Adriana Morillo Alcívar

Tutores: Cèsar Ferri Ramírez
José Hernández Orallo

Curso 2015-2016

Agradecimientos

A Dios por permitirme vivir esta experiencia, por darme la sabiduría y la fuerza necesaria para concluir con éxito esta etapa.

A mi esposo por su apoyo incondicional.

A mi familia y amigos por cada oración y cada palabra de ánimo.

A mis tutores por su acompañamiento durante la realización de este trabajo.

A mis profesores y compañeros del MUGI, por su hospitalidad, por las vivencias compartidas y por su aporte en el ámbito personal y profesional.

Resumen

Los modelos de clasificación se generan por algoritmos de aprendizaje supervisado, que aprenden a través de un conjunto de datos de entrenamiento. Estos modelos establecen relaciones entre las instancias, que les permiten predecir si pertenecen, o no, a un mismo tipo o clase. Cuando los clasificadores se usan en aplicaciones de la vida real como: discriminación de imágenes, diagnósticos en medicina, gestión de las telecomunicaciones, bioinformática, clasificación de texto, detección de fraude en transacciones financieras, etc., se enfrentan a dificultades ocasionadas por la distribución de las clases y/o por los costes de clasificar erróneamente una instancia.

Existen algunas herramientas que permiten evaluar las prestaciones de los clasificadores, una de las más usadas debido a la facilidad de su interpretación es la curva ROC, que aunque tiene asociados estadísticos que permiten seleccionar o descartar modelos de acuerdo a su desempeño, no toma en cuenta la distribución de las clases y el coste de clasificación. Para solventar estas limitaciones surgieron las Curvas de Coste.

El propósito de este trabajo es realizar un estudio de las herramientas gráficas de evaluación del rendimiento de clasificadores, dando mayor énfasis a las Curvas de Coste y métodos de selección de umbral sobre clasificadores suaves. Como resultado de este trabajo se desarrolla una librería gráfica, en el lenguaje de programación R, que incorpora estas funcionalidades. Además, se incluyen algunos ejemplos del uso de la nueva librería con conjuntos de datos reales y métodos de clasificación conocidos. Estos ejemplos ilustran las ventajas que presenta la utilización de las Curvas de Costes y los métodos de selección de umbral cuando se requiere evaluar el rendimiento de clasificadores en entornos con contextos cambiantes.

Palabras clave: aprendizaje supervisado, aprendizaje automático, clasificación, evaluación del rendimiento, curvas ROC, Curvas de Coste, lenguaje de programación R

Abstract

Classification models are generated by supervised learning algorithms that learn through a training dataset. These models establish relationships between instances, which allow them to predict whether they belong or not to the same type or class. When classifiers are used in real-life applications, such as image discrimination, medical diagnosis, telecommunications management, bioinformatics, text classification, fraud detection in financial transactions, and others, they face difficulties caused by the distribution of classes and/or the cost of misclassifying an instance.

There are some tools that can evaluate the performance of classifiers. In particular, the ROC curve is one of the most used due to its ease of interpretation. Although it has statistical methods that allow to select or exclude models according to their performance, the ROC Curve does not take into account distributions of classes and misclassification costs. The Cost Curves appeared as a solution to overcome these limitations.

This paper aims to research graphic tools for performance evaluation of classifiers, focused on Cost Curves and threshold choice methods applied to soft classifiers. As a result of this analysis, we develop, using the programming language R, a graphical library that incorporates these functionalities. We include some examples using the new library with real datasets and well-known classifiers methods. These examples illustrate the advantages that introduce the use of Cost Curves and threshold choice methods when we want to assess the performance of classifiers in environments with changing context.

Key words: supervised learning, machine learning, classification, performance evaluation, ROC curves, Cost Curves, programming language R

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

"El aprendizaje automático es la vía de transformación principal que nos está llevando a repensar todo lo que hacemos."

Sundar Pichai, CEO de Google

La ingente cantidad de información disponible en internet y el gran progreso de las máquinas en cuanto al aumento de la capacidad de memoria y cálculo, han contribuido enormemente al avance del aprendizaje automático en los últimos años y lo han convertido en «parte central de la nueva revolución tecnológica basada en el uso inteligente de la información» [1]. De forma general, el aprendizaje automático busca a través de la aplicación de algoritmos, que las máquinas puedan llegar a aprender, estableciendo patrones sobre los datos y realizando predicciones sobre los mismos. Este aprendizaje se consigue por medio de la experiencia (entrenamiento) y el establecimiento de medidas de rendimiento (*performance*).

En la actualidad existen un sin número de aplicaciones cada vez más cotidianas que se benefician del aprendizaje automático, como es el caso de detecciones de fraudes con tarjetas de crédito, reconocimiento de voz o imágenes, sistemas de recomendación de productos en tiendas online, sistemas de diagnósticos médicos, sistemas de predicción de ventas, entre otros [36].

Una tarea muy importante abordada tradicionalmente por el aprendizaje automático es el problema de clasificación o reconocimiento de patrones, que consiste en la definición de modelos o algoritmos capaces de estimar valores discretos, generalmente binarios, en función de un conjunto de variables independientes. Un ejemplo de la aplicación de un algoritmo de clasificación es la detección de spam en mensajes de correo electrónico donde se utiliza el asunto, texto, remitente y otros elementos del mensaje, para discriminar los mensajes de correo no deseado de los mensajes reales. Dado que los clasificadores permiten asistir en la toma de decisiones, la evaluación de estos algoritmos es fundamental y en ciertas ocasiones crucial.

Existen diferentes métricas de evaluación que estiman entre otras cosas el porcentaje de error o acierto en las predicciones, la precisión, etc. Estas métricas se obtienen a partir de los valores de la matriz de confusión, una matriz cuadrada de orden dos, en la que las filas representan las clases actuales de una instancia y las columnas las clases estimadas [13]. Los valores que corresponden a cada una de

las celdas de la matriz de confusión, se obtienen al aplicar un algoritmo de clasificación a un conjunto de ejemplos de prueba y contado el número de ejemplos que pertenecen a una clase u otra. Contrastando con las clases originales se obtienen: la tasa de verdaderos positivos, falsos negativos, falsos positivos y verdaderos negativos respectivamente.

Además de las métricas mencionadas existen técnicas de visualización de clasificadores en función de su rendimiento, el análisis ROC (*Receiver of Operational Conditions*) es uno de los más utilizados en el área. Su idea conceptual consiste en la representación en el plano de la tasa de aciertos (verdaderos positivos) versus la tasa de falsas alarmas (falsos positivos). Se concibió en la segunda guerra mundial para el análisis de señales de radar y posteriormente se utilizó en el análisis de respuesta de transistores y en aplicaciones de diagnóstico médico. Y fue en los noventa cuando su uso se extendió al campo de la minería de datos [8].

A pesar de que las curvas ROC resultan a simple vista sencillas de interpretar y además tienen asociados estadísticos que permiten seleccionar o descartar modelos de acuerdo a su desempeño, no toman en cuenta la distribución de las clases y el coste de clasificación. Estos dos componentes son muy frecuentes en problemas de la vida real, por lo que han sido abordados con gran interés por la comunidad científica.

La distribución de clases está relacionada con la cantidad de objetos existentes en cada una de las clases, así tenemos distribuciones balanceadas y desbalanceadas. Las balanceadas son aquellas en las que todas las clases tienen igual número de instancias en los clasificadores binarios, esto implica que el cincuenta por ciento pertenecen a una clase y el otro cincuenta por ciento a otra. En el caso de las desbalanceadas una de las clases concentra mayor número de instancias que otra o viceversa, en la práctica este tipo de distribución es mucho más habitual [28]. Trabajar con clases desbalanceadas puede causar un sesgo en los clasificadores, ya que podrían favorecer a la clase mayoritaria, realizando una mala clasificación de aquellas instancias que pertenecen a la clase minoritaria. Esto se puede observar claramente en aplicaciones de detección de fraude en transacciones bancarias, en donde, por cada trescientas mil transacciones reales se pueden encontrar cinco que son fraudulentas, como es evidente la prioridad en este caso es reconocer aquellas que pertenecen a la clase minoritaria [32].

Por otra parte, los costes de clasificación hacen referencia a las consecuencias producidas por los errores del modelo predictivo, por ejemplo, en una central nuclear el dejar cerrada una válvula cuando es necesario abrirla podría provocar una explosión, mientras que abrir una válvula cuando puede mantenerse cerrada, podría provocar una parada del sistema. A pesar de que las dos decisiones son provocadas por un error en la predicción, es indiscutible que los costes asociados a una decisión u otra difieren significativamente. Por lo tanto, lo importante no es obtener un clasificador que se equivoque lo menos posible sino que tenga un coste menor. El coste total dependerá de un contexto que a su vez estará determinado por los costes de clasificación y la distribución de clases [12].

Un planteamiento relativamente nuevo para visualizar el rendimiento de los clasificadores, que incorpora estos dos elementos (costes de clasificación errónea y distribución de las clases), son las Curvas de Coste. Estas gráficas están diseñadas especialmente para visualizar el error esperado en función de las condiciones

de operación y además conservan algunas propiedades importantes de las curvas ROC, habiendo una relación de dualidad entre ambas [6].

Las Curvas de Coste poseen ciertas características que les permiten dar respuesta a cuestiones planteadas con bastante frecuencia, tales como: cuál es el rendimiento de un clasificador dados un coste de clasificación errónea específico y unas proporciones de clase, cuál es la diferencia entre los rendimientos de dos clasificadores, para qué costes de clasificación errónea y probabilidades de clase es mejor un clasificador u otro, etc. [5].

Además, los métodos de selección de umbral (*treshold*), que resultan del análisis de las Curvas de Coste, permiten establecer umbrales adecuados a las condiciones de operación del modelo, a través de un umbral los modelos de clasificación que estiman probabilidades o *scores* se pueden convertir en clasificadores discretos, esto facilita la comparación entre diferentes modelos de clasificación.

A pesar del notable aporte que ofrecen las Curvas de Coste y los métodos de selección de umbral al campo del aprendizaje automático y concretamente a la línea de investigación sobre evaluación de clasificadores, esta temática se ha quedado limitada al ámbito teórico, debido a que hasta el momento no se cuenta con un paquete informático que explote sus potencialidades. Por lo tanto, el objetivo principal de este trabajo es ofrecer una herramienta computacional que permita que los investigadores inviertan más tiempo en el análisis de las curvas que en su construcción. Para ello se plantea el desarrollo de una librería gráfica que incluya funcionalidades para trazar las Curvas de Coste y calcular sus respectivas áreas bajo la curva, una medida importante cuando se comparan clasificadores.

El programa elegido para la implementación de la librería debido a sus altas prestaciones es el lenguaje de programación R, un entorno de software libre que brinda soporte al campo del aprendizaje automático a través una serie técnicas estadísticas y de visualización. Incluye funcionalidades para realizar pruebas estadísticas clásicas, análisis de series temporales, modelos de regresión lineal y no lineal, modelos de clasificación, agrupación, y otros.

En los últimos años el uso de R se ha extendido entre los científicos de datos (*data scientists*) y desarrolladores de software de estadística y análisis de datos. Esto se ve reflejado en la última encuesta de Rexer Analytics (la mayor encuesta efectuada anualmente a profesionales del área de la minería de datos, *data science* y análisis de datos) realizada a 1200 *data scientists*, en la que R se posiciona como una herramienta primaria líder sobre otros lenguajes de programación incluidos paquetes estadísticos licenciados como SAS, SPSS, entre otros [30]. El éxito de R radica en gran parte a que es un software gratuito altamente escalable y con una comunidad de usuarios muy amplia [35].

En base a lo expuesto, la estructura de este trabajo se compone de tres capítulos adicionales:

El segundo capítulo presenta el marco teórico que abarca conceptos generales del aprendizaje automático y definiciones más específicas empleadas para definir y construir las funcionalidades de la librería. Se enfatiza especialmente en al análisis de las Curvas de Coste y los métodos de selección de umbral.

El tercer capítulo empieza con una breve descripción de las bondades del lenguaje de programación utilizado, luego continúa con la descripción de la metodología de implementación y el desarrollo de la librería.

El cuarto capítulo presenta dos casos de uso de la librería. La finalidad de estos ejemplos es mostrar la utilidad de la herramienta y de las propiedades de las Curvas de Coste y los métodos de selección de umbral, en la evaluación del rendimiento de clasificadores en entornos con contextos cambiantes. También se muestra las prestaciones de la librería para comparar el desempeño de varios clasificadores a la vez.

Finalmente se presentan las conclusiones de este trabajo, las referencias bibliográficas y los apéndices que aportan detalles de la implementación.

CAPÍTULO 2
Aprendizaje automático

"Un programa computacional se dice que aprende de la experiencia E, respecto a algunas tareas T y un rendimiento P, si su rendimiento en la tareas T, medido por P, mejora con la experiencia E."

Tom M. Mitchell, informático

La concepción de la inteligencia está intrínsecamente conectada con la capacidad de aprender de la experiencia y de adaptarse a nuevas situaciones [17]. Así, el aprendizaje automático es una parte fundamental de la inteligencia artificial, que toma como referencia la habilidad de aprender de los sistemas biológicos y en particular de los humanos, para crear algoritmos computacionales, que consigan que una máquina pueda aprender.

En los primeros años del aprendizaje automático la mayor diferencia que se apreciaba entre el aprendizaje humano y el aprendizaje de una máquina, recaía en el hecho de que una persona podía simultáneamente ejecutar una tarea y aprender cómo mejorarla, mientras que una máquina se limitaba a ejecutar una serie de tareas pre-configuradas de manera muy eficiente pero sin la capacidad de mejorarlas, a pesar de la experiencia que le suponía la ejecución repetida de estas tareas. Esta concepción ha ido cambiando gracias al progreso de la tecnología y al surgimiento de nuevas técnicas de aprendizaje como el aprendizaje profundo, que está permitiendo que la brecha entre el aprendizaje humano y el aprendizaje de una máquina disminuya [22].

En términos generales se sigue considerando que un programa computacional aprende si y sólo si: dada una tarea, un entrenamiento, una experiencia y una métrica de rendimiento, es capaz de mejorar la tarea [3]. Por lo tanto, el aprendizaje automático busca crear sistemas que permitan que una máquina aumente su conocimiento y mejore su desempeño (*performance*), en función de su experiencia y entrenamiento previo. Esta experiencia se obtiene a través de la interacción con el entorno, la observación, las acciones y el conjunto de datos de entrada que alimenta al sistema y sirve de entrenamiento para aprender un determinado número de conceptos. En la actualidad, la información que alimenta estos sistemas puede ser generada tanto por personas como por el entorno a través del uso de sensores.

En esencia un sistema de aprendizaje se puede representar por el esquema de la Figura 2.1. Este sistema se compone de: un sistema adaptativo que permite que

una misma tarea se pueda ejecutar con mayor eficacia para la siguiente iteración y de un comparador que se encarga de entrenar o adquirir experiencia.

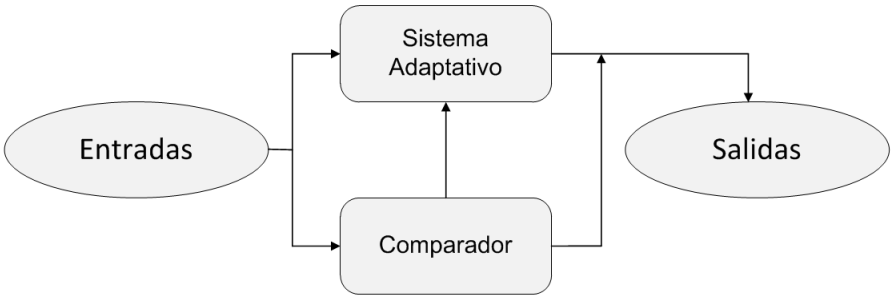


Figura 2.1: Esquema general de un sistema de aprendizaje

Debido a que el éxito de un algoritmo depende del conjunto de datos de entrada con los que realiza el entrenamiento, el aprendizaje automático está ligado a técnicas basadas en el tratamiento de datos, conceptos de ciencia computacional, estadística, probabilidad y optimización.

2.1 Tipos de aprendizaje

Dependiendo de la forma en la que los algoritmos adquieren el conocimiento, de manera general se tiene: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje reforzado.

2.1.1. Aprendizaje supervisado

Son algoritmos que se entrenan y adquieren conocimiento a partir de un conjunto de ejemplos suministrados. Las entradas del algoritmo están perfectamente definidas y etiquetadas, y se conoce de antemano la salida que se desea obtener.

La meta del aprendizaje supervisado es inferir la relación entre las entradas y salidas subyacentes, basado en las entradas y salidas de un conjunto de muestras o ejemplos. Una vez que ha conseguido aprender esta relación, debe ser capaz de predecir la salida únicamente con los datos de entrada [21]. Por ejemplo, dado un conjunto de imágenes de personas etiquetadas de acuerdo a su género, el algoritmo deberá aprender la relación existente entre la imagen y el género de la persona. La salida esperada del sistema será la predicción del género que pertenece a una nueva imagen suministrada. Los problemas clásicos abordados en esta área son: clasificación y regresión. En la (Figura 2.2 (a)) se puede observar un conjunto de puntos etiquetados ya sea por color o forma geométrica, este conjunto es la entrada típica de modelos de aprendizaje supervisado. A través de estos puntos y sus etiquetas el algoritmo deberá aprender a encontrar los patrones que describen y discriminan a los objetos (recta de color rojo Figura 2.2 (b)).

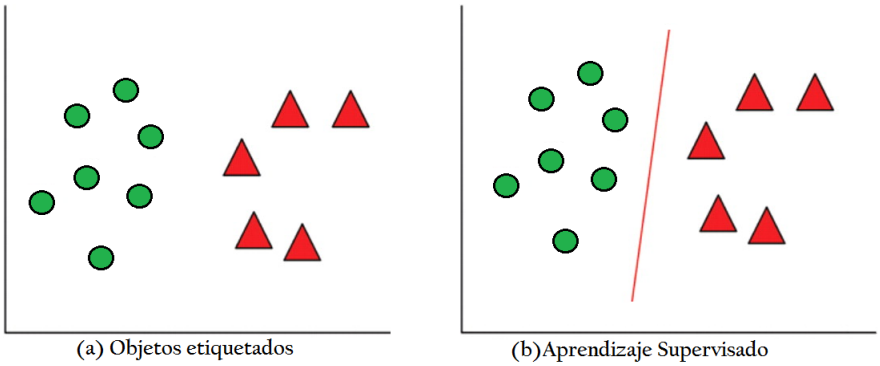


Figura 2.2: Aprendizaje Supervisado

2.1.2. Aprendizaje no supervisado

Son algoritmos que aprenden mediante exploración autónoma. En contraste con el aprendizaje supervisado los datos de entrada no están definidos y tampoco se proporcionan las salidas en el conjunto de datos de entrenamiento, se delega completamente al algoritmo la tarea de encontrar la estructura y patrones de comportamiento entre los objetos o ejemplos suministrados [29]. La meta del aprendizaje no supervisado es extraer información valiosa escondida entre los datos de entrenamiento.

De manera análoga al ejemplo anterior, la entrada del algoritmo podría ser un conjunto de imágenes de personas sin etiquetar, la salida no se específica, por tanto las agrupaciones podrían estar determinadas por patrones de edad, raza, género, etc. Un problema común abordado por el aprendizaje no supervisado es el agrupamiento o *clustering*.

La Figura 2.3(a) muestra un conjunto de puntos que se proporcionan al modelo de aprendizaje no supervisado, en este caso ninguno de los puntos tiene una etiqueta específica, por tanto, es el algoritmo que después de inspeccionar a los objetos debe determinar los patrones existentes entre ellos (Figura 2.3(b)).

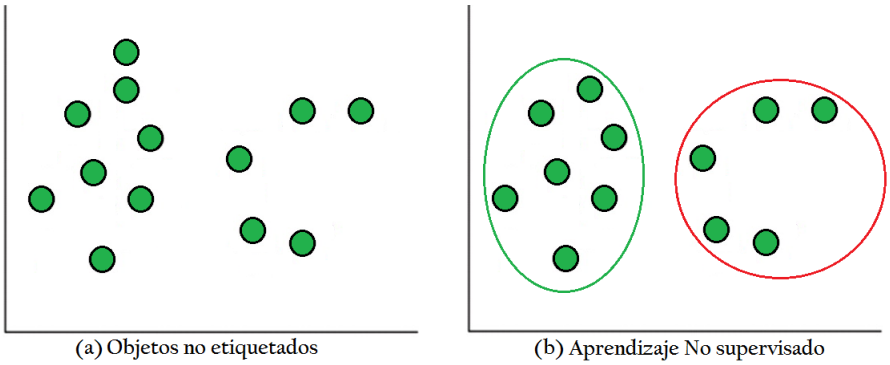


Figura 2.3: Aprendizaje no Supervisado

2.1.3. Aprendizaje con refuerzo

El aprendizaje con refuerzo es un algoritmo intermedio entre el aprendizaje supervisado y el aprendizaje no supervisado, inspirado en la sicología conductis-

ta. Los algoritmos de este tipo representan su entorno mediante estados y todas las posibles opciones de cada estado. Aprenden utilizando una función de recompensa que devuelve un conjunto de puntuaciones (*scores*), las mismas que indican cuán buenas son las consecuencias de ejecutar cada acción en cada estado. La meta del aprendizaje reforzado es lograr que la suma de recompensas de cada una de las acciones en los diferentes estados se maximice [39].

La principal diferencia con otros tipos de aprendizaje es que usa como información para el entrenamiento la evaluación de las acciones y el resultado de estas en su entorno, pero no menciona que acción es la correcta o la que se debería tomar.

2.2 Paradigmas de aprendizaje

Otro enfoque habitual del aprendizaje automático es considerar la resolución de problemas como un tipo de aprendizaje o paradigma, que consiste en que una vez resuelto un tipo de problema, el algoritmo debe ser capaz de reconocer la situación problemática y reaccionar usando la estrategia aprendida [27]. Algunos de los problemas de aprendizaje más comunes, se detallan a continuación:

2.2.1. Clasificación

Se utiliza para estimar valores discretos en función de un conjunto de variables independientes, los algoritmos de clasificación aprenden a través de un conjunto de entrenamiento en el que los ejemplos vienen etiquetados por una clase [28].

Un ejemplo muy usual es la clasificación de documentos en diferentes categorías como: economía, política, negocios, deportes, etc. En general, el número de etiquetas o categorías no suele ser muy grande, sin embargo, existen aplicaciones en las que el número se incrementa exponencialmente, tal es el caso del reconocimiento óptico de caracteres, reconocimiento de voz, clasificación de texto, entre otros.

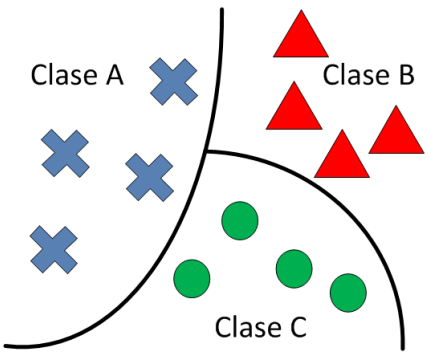


Figura 2.4: Ejemplo de Clasificación

2.2.2. Regresión

El modelo asigna a cada instancia un valor numérico real, por tanto, la salida a predecir debe ser continua. El modelo encuentra las relaciones entre las variables mediante múltiples iteraciones que se van ajustando en función de una medida de error.

Un ejemplo de este tipo de problema es un sistema de predicción del precio de una vivienda, que toma en cuenta múltiples factores, como: la superficie, el número de habitaciones, la localización, etc.

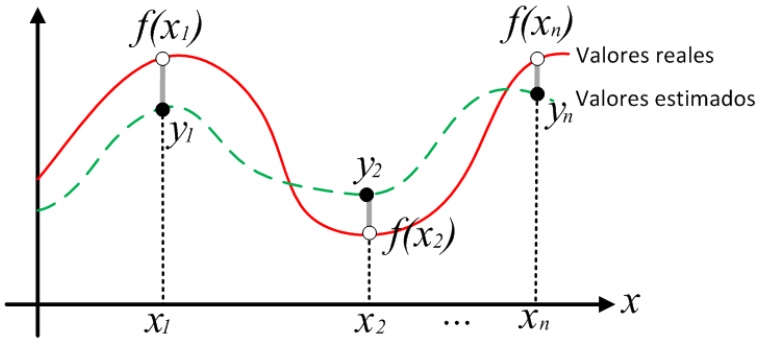


Figura 2.5: Ejemplo de Regresión

2.2.3. Agrupamiento *(Clustering)*

Es un problema típico de aprendizaje no supervisado, este algoritmo intenta encontrar patrones en la estructura de los datos que permitan agrupar o separar los objetos. El objetivo final es fragmentar el conjunto de elementos dados en regiones homogéneas.

Se pueden aplicar, por ejemplo, para clasificar cualquier tipo de artículo por temas según su contenido: deportes, ciencias, literatura, etc.

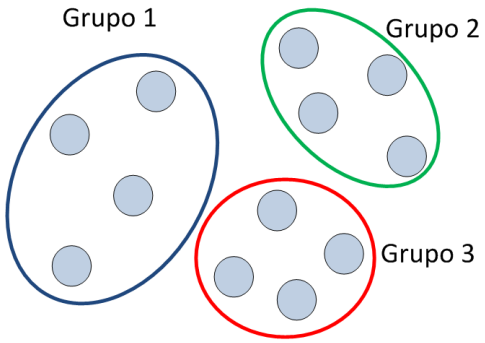


Figura 2.6: Ejemplo de Agrupamiento

2.2.4. Recomendación *(Ranking)*

Busca predecir el grado de preferencia de un elemento respecto a otros en base a una condición.

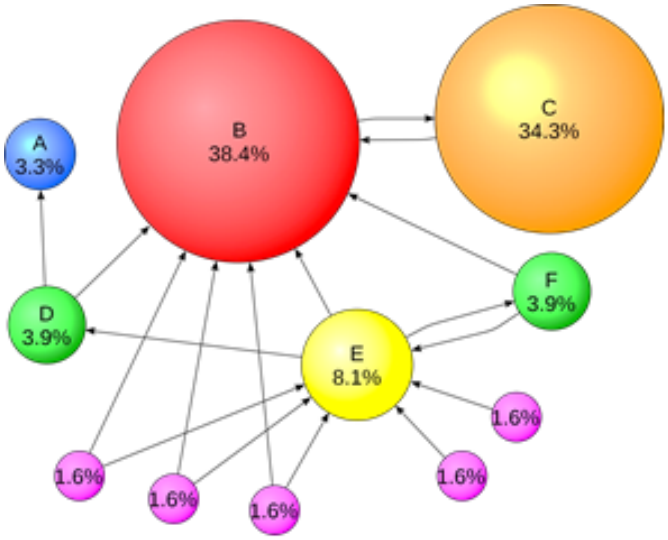


Figura 2.7: Esquema del modelo *pageRank* sobre una red simple

Como ejemplo de este tipo de problemas se puede citar la recomendación de productos para la venta online en función de compras anteriores, preferencias de artículos anteriormente visitados, históricos de compras de otros clientes e incluso de características personales, como: sexo, edad, etc. Otro ejemplo clásico de este tipo de algoritmo es la ordenación de los resultados de Google por su relevancia.

2.3 Algoritmos de clasificación

El problema de clasificación es uno de los temas más abordados en el campo del aprendizaje automático. Se trata de un problema de aprendizaje supervisado que consiste en asociar una serie de elementos (objetos, instancias, ejemplos) a unas clases determinadas. Esta asociación se realiza en base a las propiedades o atributos de los objetos.

Típicamente la entrada de un clasificador es un vector de características (patrón) con valores discretos y/o continuos y su salida teóricamente es un valor discreto denominado clase [3].

Si el clasificador tiene que decidir entre dos clases posibles, se conoce como clasificador binario. Los clasificadores que están diseñados para producir únicamente una clase de decisión (verdadero o positivo, sí o no, 1 o 0, etc.) para cada instancia se denominan clasificadores discretos o categóricos. Aquellos clasificadores que devuelven por cada instancia un valor de probabilidad o *score*, que representa el grado de pertenencia de la instancia a una clase, se conocen como clasificadores suaves, probabilísticos o estimadores de*scores*. Si los valores son estrictamente probabilidades, se deben tomar en cuenta los teoremas estándares de probabilidad, pero en el caso de que sean valores sin ninguna escala específica (*scores*) sólo se debe tomar en cuenta la condición de que: un valor de *score* alto indica una probabilidad alta de que el objeto pueda pertenecer a una determinada clase.

Algunas de las técnicas de clasificación más utilizadas se mencionan a continuación:

- Redes neuronales
- Árboles de decisión (ID3, J48 o C4.5, LMT, M5P, *Random Forest*, etc.)
- Regresión Logística
- k-means
- CN2
- CBR Razonamiento basado en casos (*Case-Based Reasoning*)
- K-NN K-vecinos más cercanos (*K-Nearest Neighbour*)
- Clasificadores Bayesianos

Además de estas técnicas existe un conjunto de algoritmos de aprendizaje automático, que aunque no son del tipo supervisado, también se utilizan para resolver problemas de clasificación. Estos algoritmos se enmarcan en lo que se conoce como aprendizaje profundo o *deep learning*, que básicamente hacen uso de redes neuronales (Figura 2.8) para la resolución de casos en los que se tienen grandes volúmenes de datos, que pueden no estar etiquetados, o estarlo parcialmente [24].

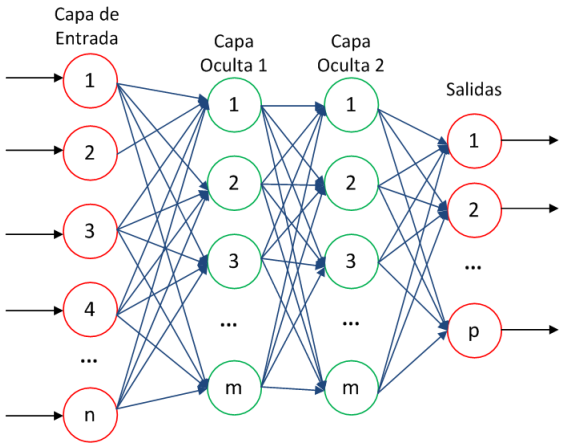


Figura 2.8: Esquema típico de un problema de Aprendizaje Profundo

En general, el desarrollo de un clasificador tiene tres fases bien diferenciadas (Figura 2.9). En la fase de entrenamiento se dota al modelo de un conjunto de datos de entrenamiento o *training dataset* (cada instancia está asociada a una etiqueta o clase) del que aprende.

Una vez que se ha definido el modelo y este ha aprendido a discriminar los objetos, se realiza la fase de evaluación. Esto consiste en probar el modelo con un conjunto de datos de prueba o *testing dataset* (las instancias no se asocian a una clase, aunque se conocen cuales son) con el que se busca determinar el rendimiento del clasificador.

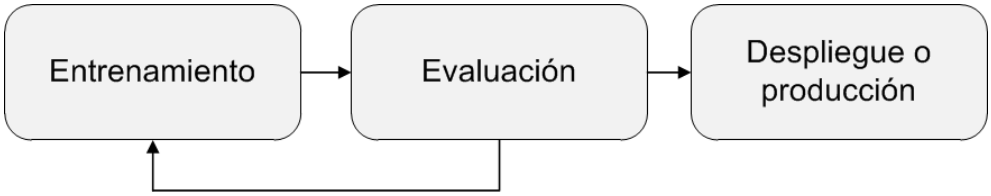


Figura 2.9: Esquema general del desarrollo de un clasificador

La fase de producción o despliegue sucede luego de que el clasificador ha pasado la fase de evaluación con un nivel aceptable de rendimiento. En este caso se trabaja con un conjunto de datos no validados (no se conocen las clases reales de las instancias).

La etapa que atañe el presente trabajo corresponde a la fase de evaluación, concretamente al análisis del rendimiento de clasificadores binarios. Por tanto, se precisa definir la notación y conceptos que se utilizarán a lo largo del estudio: Dado un clasificador binario, el espacio de salida estará denotado por , donde 0 corresponde a las clases positivas y 1 corresponde a las clases negativas $Y = 0, 1$. Los ejemplos o instancias son tomados de un espacio de instancias denotado por X. Mientras que los elementos de X y Y se pueden expresar como x y y respectivamente.

Un clasificador categórico es una función que asigna para cada objeto una clase. Un modelo o clasificador suave es una función $m : X \rightarrow R$ que asigna a los ejemplos puntuaciones o *scores* en una escala no especificada.

Con el fin de realizar predicciones en el dominio Y, el clasificador estimador de *scores* se puede convertir en un clasificador categórico mediante la fijación de un umbral de decisión t sobre los *scores*. Dado un *score* estimado $s = m(x)$, la instancia x se clasifica en la clase 1 si $s > t$ y en la clase 0, en el caso contrario.

La tasa de clases positivas o clases 0 correctamente clasificadas en función del umbral de decisión se define como $F_0(t)$ o TP(t). Así mismo, la tasa de clases negativas o clases 1 clasificadas de forma incorrecta se denota como $F_1(t)$ o $FP(t)$.

La proporción de clases se denota como π_k, para k. Donde, π_0 es la proporción de instancias clasificadas en la clase positiva y π_1 es la proporción de instancias clasificadas en la clase negativa. Si $\pi_1 = \pi_0$ se dice que las clases están balanceadas, caso contrario las clases se consideran desbalanceadas.

Las condiciones de operación (o también llamado contexto) se definen por la proporción de clases y los costes de clasificación erróneos sobre los objetos.

2.4 Evaluación del rendimiento de un clasificador

Todo algoritmo de clasificación está sujeto a la posibilidad de cometer errores en sus resultados. Por lo tanto, la evaluación del rendimiento de un clasificador es crucial para conseguir resultados aceptables una vez que el clasificador se ponga en producción.

Una primera aproximación de la evaluación del modelo consiste en aplicar el clasificador a los propios ejemplos proporcionados en el entrenamiento. Sin em-

bargo, esta estimación genera resultados muy sesgados y conlleva a errores de sobre-ajuste [45]. Para obtener resultados más realistas, se utiliza un conjunto de datos en el que se proporcionan nuevos ejemplos que no se han incluido en el proceso de entrenamiento, para ello se divide el conjunto de datos en dos subconjuntos: uno con los datos para el entrenamiento y otro con los datos para la prueba.

Si el *dataset* es muy pequeño, el número de instancias disponibles para entrenar el modelo quedaría muy reducido, esto conllevaría a obtener peores resultados. Para solventar este inconveniente se utilizan técnicas de validación cruzada o *cross-validation* que proporcionan mayor objetividad a las pruebas. La técnica generalizada de *cross-validation* se conoce como *k-fold* la descripción de su funcionamiento y la de sus variantes (*2-fold* y *Leave one out*) se describen en la Tabla 2.1 [20].

Parámetro	Descripción
2-fold	Se divide el *dataset* en dos subconjuntos de tamaños iguales S_1 y S_2. Se utiliza el subconjunto S_1 para entrenar el modelo y luego se lo evalúa con el subconjunto S_2 . Luego se repite el proceso utilizando S_2 en el entrenamiento y $S1$ en la prueba. Se elige el clasificador que tenga mejor precisión o menor error.
k-fold	Los n datos se parten k veces $(k < n)$ en 2 subconjuntos (normalmente n/k para el *test* y $n - n/k$ para entrenamiento, excluyendo cada vez un subconjunto distinto). El algoritmo se ejecuta k veces y luego se elige el clasificador con mejores resultados.
Leave one out	Sigue el mismo proceso que en el *k-fold* pero para $k = n$.

Tabla 2.1: Técnicas de validación cruzada

La media de la métrica de rendimiento obtenida de los k casos proporciona una estimación para saber cómo se va a comportar el modelo utilizando nuevos datos.

Una vez que se ha aplicado el modelo a varios *datasets* de prueba, se pueden obtener una serie de métricas que proporcionan mayor información acerca del rendimiento del clasificador.

El proceso de evaluación en la fase de desarrollo del clasificador es un proceso iterativo. Esto implica que por cada cambio realizado en el modelo, este último, se debe reevaluar con la finalidad de determinar el impacto sobre el rendimiento. Al finalizar la fase de evaluación es importante determinar qué clasificador

ha logrado un nivel aceptable de rendimiento y a su vez representa una mejora considerable frente a otros clasificadores existentes.

2.4.1. Matriz de confusión y medidas escalares

La matriz de confusión o también llamada matriz de contingencia es una matriz cuadrada de orden 2, donde, las filas representan las clases actuales de una instancia y las columnas las clases predichas o estimadas. Es una forma muy simple pero muy efectiva de visualizar los aciertos y los errores que ha cometido el clasificador.

		Clases Estimadas	
		P	N
Clase Actual	Y	Verdaderos Positivos (TP)	Falsos Positivos (FP)
	N	Falsos Negativos (FN)	Verdaderos Negativos (TN)
	Total	Instancias positivas	Instancias negativas

Figura 2.10: Matriz de Confusión

Dado un clasificador y una instancia, existen cuatro posibles salidas que se verán reflejadas en la matriz de confusión (Figura 2.10):

- Si la instancia es positiva y se clasifica como positiva, se cuenta como un Verdadero Positivo (*TP True Positive*).
- Si por el contrario se clasifica como negativa, se considera como un Falso Negativo (*FN False Negative*).
- Si la instancia es negativa y se clasifica como negativa es contada como un Verdadero Negativo (*TN True Negative*).
- En caso de que se clasifique como positiva, se suma a la cuenta de los Falsos Positivos (*FP False Positive*).

La suma de los valores de la diagonal principal representa las decisiones acertadas del clasificador, mientras que los números de la diagonal secundaria representan las decisiones equivocadas del clasificador (errores).

Alrededor de la matriz de confusión se pueden calcular varias medidas de rendimiento, las mismas que se resumen en la Tabla 2.2 [13].

Métrica	Formulación
Tasa de falsos positivos o error negativo	$FPR = \frac{FP}{TN + FP}$
Tasa de verdaderos positivos, exactitud positiva , *(sensitivity, recall o positive accuracy)*	$TPR = \frac{TP}{TP + FN}$
Tasa de verdaderos negativos, especificidad o exactitud negativa *(negative accuracy)*	$TNR = \frac{TN}{TN + FP}$
Tasa de falsos negativos o error positivo	$FPR = \frac{FN}{TP + FN}$
Precisión o valor de predicción positiva	$PPV = \frac{TP}{TP + FP}$
Valor de predicción negativa	$NPV = \frac{TN}{TN + FN}$
Promedio macro *(macro-average)*	$media(TPR, TNR)$
Break-even	$\frac{(precisión + recall)}{2} = \frac{PPV + TPR}{2}$
Medida de Fisher *(F-measure)*	$\frac{(precisión + recall)}{break - even} = \frac{2(PPV)(TPR)}{(PPV + TPR)}$

Tabla 2.2: Métricas para evaluar el rendimiento de un clasificador

Además de las métricas escalares detalladas en la Tabla 2.2, existen algunas herramientas gráficas que sirven para visualizar los rendimientos de un clasificador. Dos de ellas (curvas ROC y Curvas de Coste), se estudian con más detenimiento en las siguientes secciones.

2.4.2. Curvas ROC

El análisis ROC o curvas ROC se introduce en el campo de la minería de datos y el aprendizaje automático a finales de los 80's por Spackman (1989)[6], el mismo que demostró la potencialidad de las curvas para evaluar y comparar algoritmos. En los últimos años el análisis ROC se ha usado con mayor frecuencia en el campo del aprendizaje automático. Esto se debe a que las métricas escalares como: la precisión, *accuracy*, *recall*, etc., no son suficientes para representar el rendimiento de un clasificador [26].

Las curvas ROC son gráficas bidimensionales que trazan la tasa de verdaderos positivos versus la tasa de falsos positivos, es decir, representan la compensación entre los beneficios (TP) y los costes (FP).

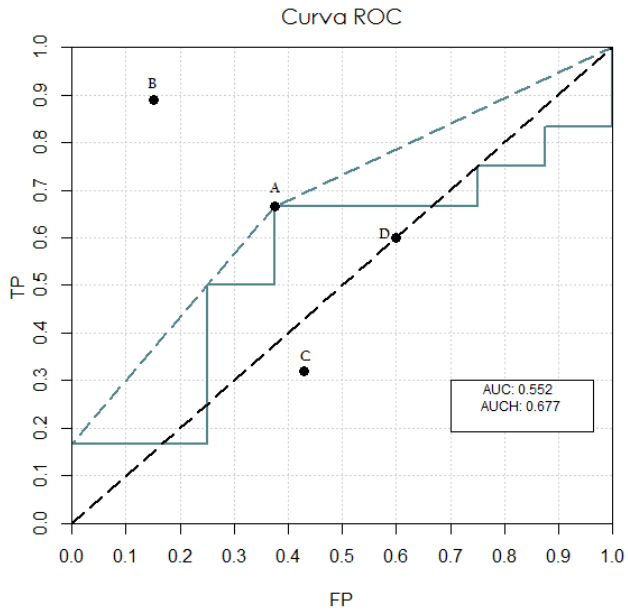


Figura 2.11: Curvas ROC

Un clasificador discreto producirá un par de valores de TP y FP que se representan como un punto en el plano, determinado por la tasa de *true positives* y la tasa de *false positives*, conocido como espacio ROC [7]. Cada punto (TP, FP) corresponde a un umbral de decisión particular, por lo que para conseguir una curva monótona se realiza un barrido de todos los posibles valores de umbral de decisión. En la Figura 2.9 se muestra el rendimiento de cuatro clasificadores discretos A, B, C y D, que corresponde a cuatro puntos en el espacio ROC. La línea azul continua corresponde a la curva ROC de un clasificador de estimación de *scores* (ejemplo tomado de [8]), esta curva se construyó utilizando umbrales de decisión sobre un conjunto de pruebas con iguales proporciones de clases. Cada punto de la curva ROC corresponde a un par ordenado (TP, FP), procedente de la aplicación un determinado umbral.

Algunos clasificadores de tipo *ranking* o estimadores de *scores*, se pueden usar junto al umbral, para producir clasificadores discretos binarios. De modo que, si la salida del clasificador está por debajo de un umbral, el clasificador produce un valor verdadero y en el caso contrario un valor falso. Cada valor del umbral producirá un punto diferente en la curva ROC.

Del análisis ROC, se derivan algunas observaciones importantes:

- El punto de coordenadas (0, 0), representa la estrategia en la que no se consideraron las clasificaciones positivas. Esto significa que no se encontraron falsos positivos pero tampoco verdaderos positivos.

- La estrategia opuesta, está representada por el punto (1,1). En este caso se consideró de forma incondicional las clasificaciones positivas.

- El punto (0,1) representa la clasificación perfecta, es decir, que el rendimiento del clasificador es del cien por ciento.

- La diagonal $y = x$, es equivalente a clasificar una instancia de forma totalmente aleatoria. Si se trata de un clasificador discreto con sólo dos clases, la probabilidad de que se clasifique a las instancia como positiva es igual a la probabilidad de se clasifique como negativa, y esto a su vez es igual a 0.5. El punto que representa este clasificador en el espacio ROC está dado por el par ordenado (0.5, 0.5).

- Informalmente un punto ROC es mejor que otro si está más alejado del eje x y más cercano al eje y, es decir, que la tasa de verdaderos positivos sea alta y a su vez que la tasa de falsos positivos sea baja. Si se analiza únicamente el clasificador A, se puede decir que la mayor *accuracy* se producirá en el punto (0.375, 0.66), cuando el umbral sea mayor o igual a un valor determinado. En el ejemplo de la Figura 2.9, el clasificador B tendría el mejor rendimiento.

- Las curvas ROC no solo muestran qué clasificador tiene mejor rendimiento promedio, sino que también, permiten visualizar las regiones en las que un clasificador es superior a otro.

- Los clasificadores cercanos tanto al eje y, como al eje x, se suelen considerar clasificadores conservadores, esto quiere decir, que realizan predicciones positivas sólo si tienen evidencias fuertes de que lo sean. Por lo que su tasa de error es muy baja al igual que su tasa de aciertos.

- Los clasificadores que se encuentran en la parte superior derecha del espacio ROC se conocen como clasificadores liberales, ya que realizan predicciones positivas con muy poca evidencia. Esto implica que la mayoría de las clasificaciones positivas son correctas, pero también tienen una alta tasa de falsos positivos.

- La envolvente convexa de la Curva ROC sirve para identificar los umbrales de decisión óptimos (línea azul entrecortada de la Figura 2.11), que conllevan a determinar potenciales clasificadores óptimos.

- Las pérdidas por errores en la clasificación no se visualizan de forma directa en las curvas ROC, pero se pueden inferir de los valores resultantes de las tasas de verdaderos y falsos positivos y de las condiciones de operación.

- Los costes de clasificaciones erróneas, así como la distribución de clases, se representan como rectas isométricas en el espacio ROC [11].

Un punto importante a considerar en las curvas ROC es que miden la habilidad del clasificador para puntuar las instancias positivas respecto a las instancias negativas. Por lo tanto, el clasificador no necesita estimar *scores* calibrados, tan solo requiere producir *scores* relativos precisos que sirven para discriminar las instancias positivas de las negativas. Una consecuencia de utilizar *scores* relativos es que las puntuaciones del clasificador no se pueden comparar a través de las clases del modelo.

Otra propiedad importante que poseen las curvas ROC, es que son insensibles a los cambios en la distribución de clases [42]. Es decir, que aunque la proporción de instancias positivas y negativas cambien en el conjunto de prueba, la curva ROC no sufrirá ningún cambio. Esta percepción no siempre resulta muy realista.

Área bajo la curva (*AUC: Area under curve*)

Cuando se comparan diferentes clasificadores, es necesario reducir el espacio ROC a una medida escalar. Un método común es calcular el área bajo la curva ROC denominada AUC, esta es equivalente a la probabilidad de que el clasificador devuelva un *score* más alto a una instancia positiva elegida al azar, que a una instancia negativa también elegida al azar [10].

Debido a que el AUC es una porción de un cuadrado de área unitaria, los valores que puede tomar están acotados en el intervalo $[0, 1]$. El clasificador aleatorio que produce la línea diagonal entre (0,0) y (1,1), tendrá un AUC igual a 0.5, por tanto un clasificador no realista deberá tener una AUC menor que este valor. Sin embargo, es posible que un clasificador con un AUC alto, tenga peor rendimiento en algunas regiones del espacio ROC que un clasificador con un valor de AUC bajo.

2.4.3. Curvas de Coste

El planteamiento de las Curves de Coste es relativamente nuevo, surgió a principios del 2000 [4], como una técnica gráfica mejorada respecto a las curvas ROC, que propone un nuevo método de visualización del rendimiento (específicamente de la tasa de error y el coste esperado) de clasificadores binarios. Para ello, toma en cuenta el rango completo de posibles distribuciones de clase y los costes ocasionados por clasificaciones erróneas. Estas dos consideraciones se conocen como condiciones de operación.

La principal ventaja que presentan frente al análisis ROC, es que permiten visualizar varios tipos de rendimiento que no se pueden apreciar a simple vista en las curvas ROC, como por ejemplo: el intervalo de confianza de un clasificador, el significado estadístico de la diferencia entre dos clasificadores, la afectación de clases no balanceadas, etc. [5].

La idea conceptual de las Curvas de Coste consiste en trazar en el eje de las ordenadas el error esperado o pérdida versus la probabilidad de coste en el eje de las abscisas. En una primera aproximación, se asume que los costes de clasificar erróneamente las instancias positivas son iguales a los costes de clasificar erróneamente las instancias negativas y estas a su vez, son iguales a 1. También se considera que $p(+)$ es el porcentaje de instancias positivas z (sesgo) cuando el clasificador está en producción. Como no necesariamente se conoce esta información cuando se realiza el entrenamiento y la evaluación, se toman en cuenta todos los posibles valores que podría tomar $p(+)$ [6].

Los valores extremos en el eje x representan el escenario en el que todos los ejemplos que se han aplicado al clasificador, están en una misma clase. Así, $x = p(+) = 0$ significa que todas las instancias son negativas y $x = p(+) = 1$ significa

que todas las instancias son positivas. En el primer caso x=0, la tasa de error se simplifica y es equivalente a la tasa de error negativa, mientras que en el segundo caso x=1 la tasa de error es igual a la tasa de error positiva.

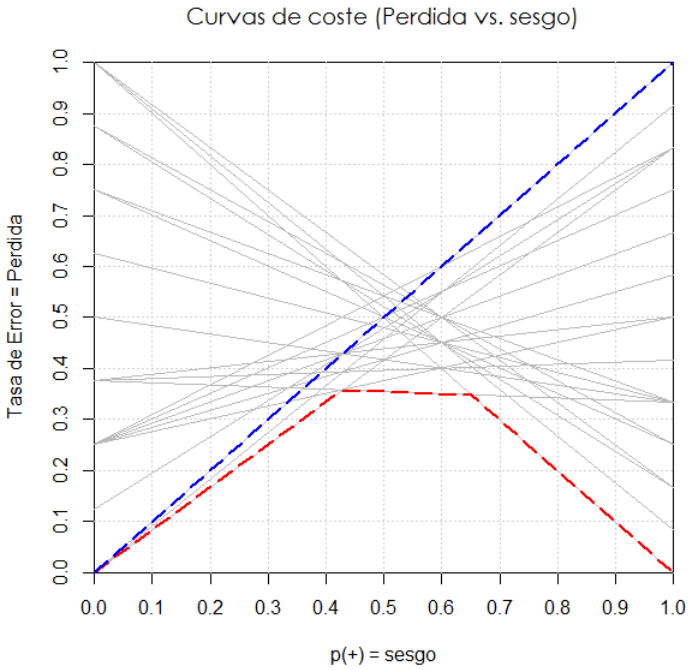


Figura 2.12: Curva de Coste normalizada

La formulación de las Curvas de Coste bajo estas circunstancias, está dada por la siguiente expresión:

$$Q_{(sesgo=skew)} = tasa\ de\ error\ o\ perdidas = (FN - FP)z + FP \tag{2.1}$$

Esta ecuación representa la curva de Coste Normalizada o Curva de Coste en función de las pérdidas y el sesgo. Las líneas de color gris en la Figura 2.11, se denominan líneas de Coste y representan la relación lineal entre el error o pérdida y el sesgo para una condición de operación específica. El máximo valor esperado de error es 1 y se produce cuando todas las instancias se clasifican de forma incorrecta.

Para generalizar la expresión de las Curvas de Coste (Figura 2.12), se deben eliminar los supuestos realizados en la primera aproximación. Así, considerando la proporción de instancias negativas π_1, la proporción de instancias positivas π_0 y la probabilidad de coste o simplemente proporción de coste c, la nueva fórmula que define las Curvas de Coste en función de las pérdidas y el coste se reescribe como:

$$Q_{(coste)} = Perdidas\ o\ Error\ esperado = 2c\pi_0(1 - TP) + (1 - c)\pi_1 FP \tag{2.2}$$

El valor máximo del error de esta curva será mayor o igual a uno. El área bajo las Curvas de Coste determina el valor de pérdida esperado.

Matemáticamente las curvas ROC y las Curvas de Coste están muy relacionadas entre sí. Concretamente existe una dualidad punto - línea entre ambas [6],

esto significa que cada punto (FP, TP) en el espacio ROC está representado por una línea en el espacio de coste y viceversa.

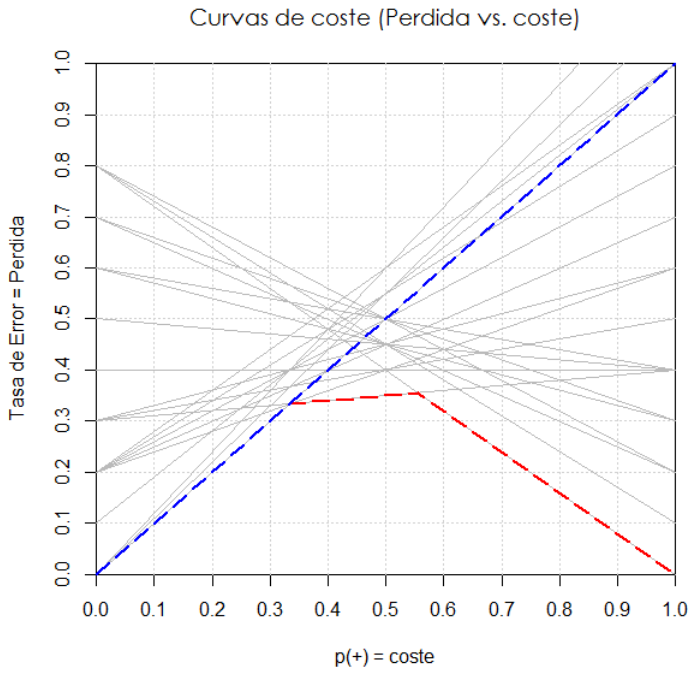


Figura 2.13: Curva de Coste no normalizada

Algunas de las implicaciones de la dualidad entre las curvas ROC y las Curvas de Coste se menciona a continuación:

- Una única matriz de confusión, que produce un único punto en el espacio ROC, también producirá una sola línea de Coste en el espacio de Coste. Esta línea de Coste atravesará por los puntos $(0, 2\pi_1 FP)$ y $(1, 2\pi_0 FN)$. Para el caso de la curva normalizada se asume que $\pi_0 = pi_1 = 0{,}5$, y por lo tanto, los puntos se reducen a $(0, FP)$ y $(1, FN)$.
- El conjunto de puntos que se utilizan para graficar la curva ROC, es equivalente al conjunto de líneas de Coste que se requieren para graficar la Curva de Coste (líneas en color gris de la Figura 2.11).
- Al igual que las curvas ROC se construyen por tramos utilizando un conjunto de puntos, las Curvas de Coste se construyen por partes usando un conjunto de líneas. La menor envolvente es una manera de construir una Curva de Coste tomando un conjunto de líneas de Coste, que se define como el menor valor de y, en alguna de las líneas de coste dadas a x.
- La menor envolvente de las Curvas de Coste es también la contraparte de la envolvente convexa de las curvas ROC.
- Los vértices de la menor envolvente en el espacio de Coste, corresponde a un segmento de línea en la envolvente convexa de la curva ROC.
- El área bajo la Curva de Coste abreviado como AUCC para evitar ambigüedades con el área bajo la curva ROC, establece el valor promedio de pérdidas esperado, tomado en cuenta el rango de operación completo.

2.5 Curvas de Coste y métodos de selección de umbral

Cuando un modelo de clasificación se aplica a un conjunto de datos cualquiera, las condiciones de operación o contexto podrían ser diferentes a las usadas en el entrenamiento. De hecho, un modelo puede ser usado en varios contextos, con diferentes resultados. Un contexto puede significar diferentes proporciones de clases, diferentes costes sobre los ejemplos o algunos otros errores que se generan al aplicar el modelo y que podrían ser relevantes [15].

Generalmente, los costes por clasificaciones correctas se establecen en cero. Esto significa que para clasificadores binarios pueden tener una matriz de costes con dos valores $c_k \geq 0$, representado los errores de clasificación erróneos de una instancia de clase k. Para normalizar los costes se estableció $b = c_0 + c_1$ y $c = \frac{c_0}{b}$, donde c se refiere a la proporción de coste. Si $b = 2$, la pérdida es conmensurable a la tasa de error (asumiendo $c_0 = c_1 = 1$).

Las pérdidas que se producen para un valor de umbral t y una proporción de coste c, está dada por la siguiente función:

$$Q_{coste}(t;c) = 2\{c\pi_0(1 - TP(t)) + (1 - c)\pi_1 FP(t)\} \tag{2.3}$$

Si interesa analizar la proporción de clases y la proporción del coste al mismo tiempo, la fórmula anterior queda definida por la ecuación:

$$Q_{sesgo}(t;z) = z(1 - TP(t)) - (1 - z)FP(t) \tag{2.4}$$

Donde z (proporción de clases y proporción del coste) se define por la siguiente expresión:

$$z = \frac{c_o\pi_0}{c_0\pi_0 + c_1\pi_1} = \frac{c_o\pi_0}{c\pi_0 + (1 - c)(1 - \pi_0)} \tag{2.5}$$

El umbral de decisión cobra mayor importancia cuando se comparan dos clasificadores utilizando las curvas ROC, ya que representa el punto de cruce entre esas dos curvas. A partir de este punto uno de los clasificadores presentará mejor rendimiento que otro y viceversa. Si se puede determinar el punto de corte y además se puede visualizar directamente para cada valor de coste su respectivo valor de pérdida, se podrá seleccionar el clasificador que mejor se comporte para determinadas condiciones de operación [8].

Una cuestión clave en la aplicación de un modelo de varias condiciones de operación es cómo elegir el umbral de decisión en cada uno de ellos. En el caso de un clasificador categórico, esta pregunta se responde fácilmente. Sin embargo, en el caso general, cuando se trabaja con un modelo suave o clasificador probabilístico, se tiene que decidir cómo establecer el umbral. La idea fundamental es la noción de un método de elección de umbral T, que dada una condición de operación (proporción de coste $T(c)$ o sesgo $T(z)$ devuelva un valor de umbral apropiado para el modelo [15].

Hay varias opciones razonables para definir la función T. Se puede establecer un umbral fijo para todas las condiciones operativas; se puede establecer el umbral observando la curva ROC (o su envolvente convexa) y usando la proporción de coste o el sesgo para intersecar la curva ROC; se puede establecer un umbral mirando los *scores* estimados, sobre todo cuando representan probabilidades; o se puede establecer un umbral independientemente del *ranking* o de los *scores*. La forma en que se fija el umbral puede afectar drásticamente el rendimiento del clasificador.

Cada método de selección de umbral dará como resultado una nueva curva de coste.

2.5.1. *Test optimal*

El método de selección de umbral, *Test optimal*, se basa en la presunción optimista de que se tiene la información completa sobre las condiciones de operación, cuando el clasificador está en producción. Por lo tanto, dadas: la proporción de clases y la proporción de costes, se puede elegir un umbral que minimice la pérdida utilizando el modelo actual [14]. Generalmente, esto ocurre cuando se usa el mismo conjunto de datos para el *training* y para el *test*. Este método se denota como: T_c^O y está dado por la siguiente expresión:

$$T_c^O \triangleq \underset{t}{argmin}\{Q_{coste}(t;c)\} \tag{2.6}$$

De manera similar, si se define en función del sesgo, se tiene:

$$T_z^O \triangleq \underset{t}{argmin}\{Q_{sesgo}(t;c)\} \tag{2.7}$$

La Curva de Coste que resulta de aplicar este método de selección de umbral se conoce como la menor envolvente y se define como el menor valor de pérdida para cualquier valor de c o z dado. En otras palabras la menor envolvente represente el rendimiento óptimo del clasificador en cualquier condición de operación. Cada segmento de la menor envolvente corresponde a un punto de la envolvente convexa de la curva ROC. Las curvas trazadas con líneas entrecortadas y de color rojo de las Figuras 10 y 11, son un ejemplo de las Curvas de Coste que se obtienen utilizando este método de selección de umbral.

2.5.2. *Train optimal*

Este método de selección parte de la misma idea del método *Test optimal*, pero en vez de asumir que se conocen las condiciones de operación en la fase de despliegue del clasificador, se utilizan las condiciones de operación empleadas en la fase de entrenamiento. En la Figura 2.14 ((a) en función del coste y (b) en función del sesgo), la curva de color verde y línea continua es el resultado de utilizar el método de selección de umbral *Train optimal* y de aplicar el modelo a un *dataset* que no ha sido utilizado en el entrenamiento. Las condiciones de operación se obtienen de la Curva de Coste de menor envolvente (línea azul entrecortada). Si

la curva *Train optimal* difiere demasiado de la curva *Test optimal*, el modelo no funcionará adecuadamente en la fase de producción.

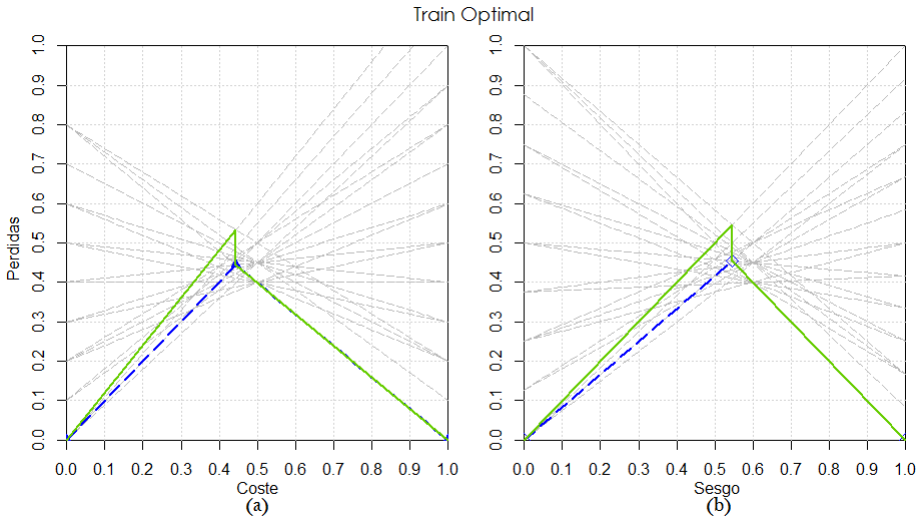


Figura 2.14: Curva de Coste - Método de selección de umbral *Train optimal*

2.5.3. *Score driven*

El *Score driven* es un método de selección probabilístico, que asume que los *scores* del clasificador son probabilidades de clases entre 0 y 1, por lo que se considera como una forma natural de selección del umbral. Esto implica que, dado una condición de operación, una instancia se puede predecir positiva si el *score* es mayor que cero y negativa en el caso contrario [9]. Con esta simple regla de decisión se pueden visualizar la pérdida para cualquier condición de operación en el espacio de coste.

Este método de selección de umbral es particularmente sensible a la estimación de las probabilidades. Si las probabilidades estimadas están altamente concentradas (p. ej. si la mitad de las instancias están en el rango [0.3, 0.4]) y se usa una probabilidad en este rango como un valor de umbral (p. ej. igual a 0.35) una mínima variación en la estimación de las probabilidades podría cambiar drásticamente las predicciones y en consecuencia el error esperado. Este problema también afecta al método de selección de umbral *Test optimal*, debido a que se puede estimar el umbral óptimo en una curva ROC trazada con un conjunto de datos validado y luego tomar el *score* (probabilidad estimada) que conduce a la elección óptima. Por lo que, el método *Score driven* y el método *Test optimal* son equivalentes cuando el modelo está perfectamente calibrado.

Formalizando la función de selección de umbral definida por la condición de operación y la proporción de coste, se tiene:

$$T(c) \triangleq c \tag{2.8}$$

Si las condiciones de operación están definidas por el sesgo, el umbral se determina por la ecuación:

$$T(z) \triangleq z - T(c)\frac{z}{c} \tag{2.9}$$

El uso de esta función de selección del umbral produce una nueva curva denominada Curva Brier (Figura 2.15). Esta curva está atravesada por las líneas de coste y no podrá estar por debajo de la Curva de Coste de menor envolvente. Las Curvas Brier muestran el rendimiento para todo el rango posible de condiciones de operación, esto significa, que puede funcionar de manera similar al análisis ROC, pudiendo descartar clasificadores dependiendo de una condición de operación. Y además combinando clasificadores de modo que se obtenga un valor mínimo de pérdida.

Una característica importante de este método de selección de umbral es que el valor del área bajo la Curva Brier coincide con el valor del *Brier score* o error cuadrático medio del clasificador.

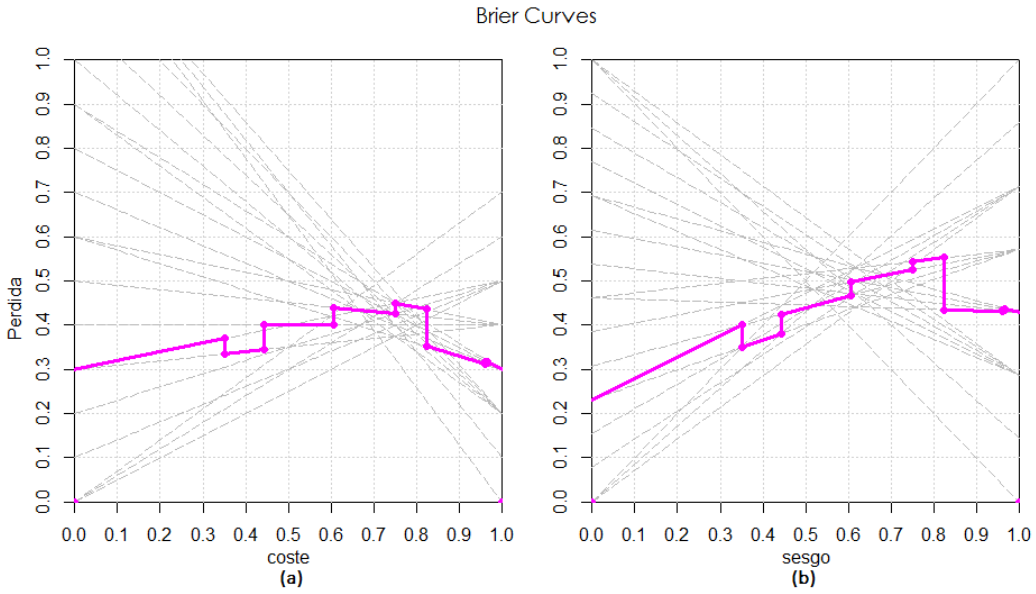


Figura 2.15: Curva de Coste - Método de selección de umbral *Score driven*

2.5.4. *Rate driven*

El método de selección de umbral *Rate driven*, considera la proporción de positivos que se quiere predecir. Si se encuentra un punto de la curva ROC (trazado con el conjunto de datos del entrenamiento o un conjunto de datos validado) que interesa utilizar para establecer el umbral, solo se tiene que calcular la tasa de positivos estimados (proporción de predicciones positivas) y usarla como referencia para el conjunto de datos de producción. La única limitación al utilizar este tipo de ratios en vez de un *score* numérico, es que, únicamente cobran sentido si se tiene una batería de predicciones. Este último escenario es una situación muy común [15].

El *recall* o tasa de positivos estimados (abreviado como ratio), está definida como: $R(t) = \pi_0 F_0(t) + \pi_1 F_1(t)$ y en función del sesgo se tiene $R_z(t) = \frac{(F_0(t)+F_1(t))}{2}$. La siguiente expresión establece el umbral con el cual se consigue un ratio igual a las condiciones de operación.

$$T^{rd}_{cost}(c) \triangleq R^{(-1)}(z) \tag{2.10}$$

Análogamente, para el sesgo:

$$T^{rd}_{skew} \triangleq R^{(-1)}_z(z) \tag{2.11}$$

El método *Rate driven* es un método natural para elegir los valores de umbral, especialmente cuando sólo se cuenta con un ranking o un clasificador probabilístico calibrado pobremente.

La curva que se produce aplicando el umbral resultante de este método de selección, se denomina Curva *Rate driven* (Figura 2.16), y su área bajo la curva es igual a $\pi_1\pi_0(1-2AUC)+\frac{1}{3}$ en el caso de sesgos uniformes es igual a $(1-2AUC)\frac{1}{4}+\frac{1}{3}$.

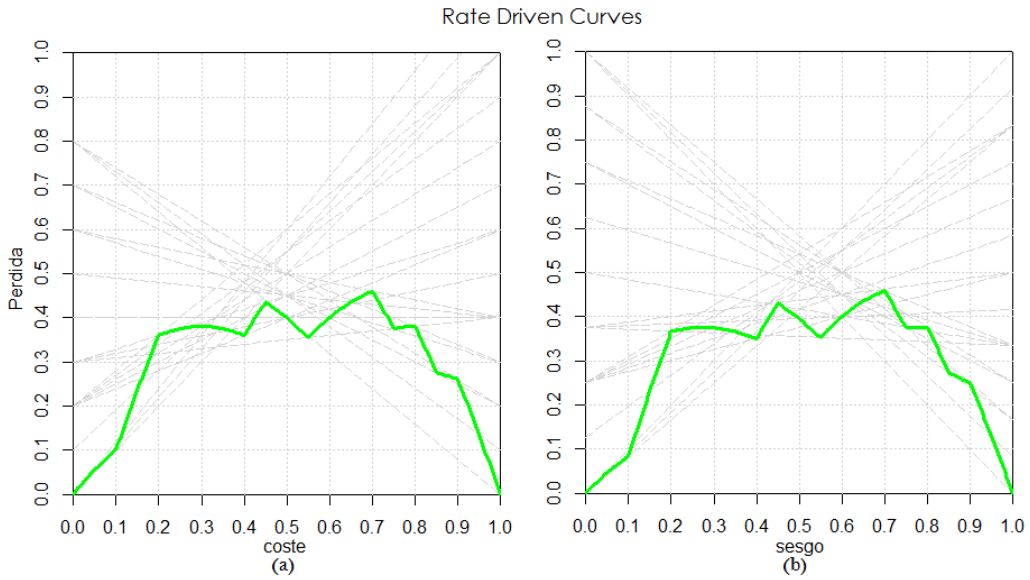


Figura 2.16: Curva de Coste - Método de selección de umbral *Rate driven*

Curvas Kendall

Las curvas *Rate driven*, se pueden descomponer en dos curvas: por un lado se tienen las Curvas de Coste *Rate driven* para un ranker perfecto Q^*_{cost} y por otro lado las curvas Kendall Q^τ (Figura 2.17).

Dado que la descomposición es puntual para cada proporción de coste c, las curvas Kendall se pueden interpretar como la pérdida esperada de clasificación debido al rendimiento del clasificador. Es decir, que para cada valor de c, las curvas Kendall muestran la pérdida esperada del modelo, una vez que se ha descontado las pérdidas del ranker perfecto. Esta última pérdida es compartida por todos los modelos [15].

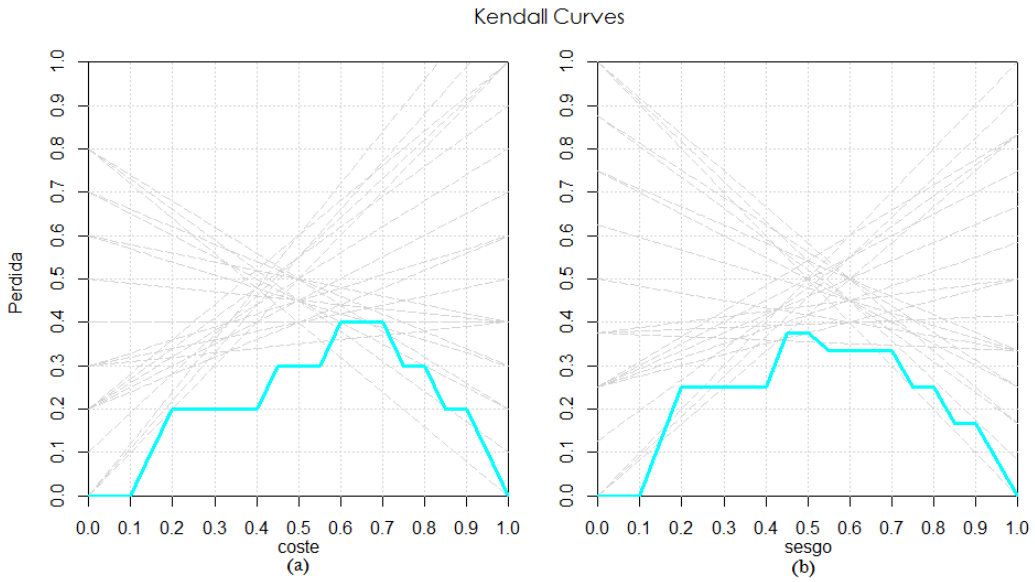


Figura 2.17: Curva de Coste - Método de selección de umbral *Curvas Kendall*

Es importante resaltar que las curvas Kendall son el resultado de la diferencia de dos curvas ($Q^{\tau} = Q_{cost} - Q^{*}_{cost}$), por lo que no es una Curva de Coste en sí misma. Esto implica que no se intersequen con las líneas de Coste como lo hacen las curvas *Rate driven*.

La curva Kendall muestra el número de pares discordantes que se distanciarán del ranking perfecto, denotado por $K_{\tau} = \pi_0 npi_1 n(1 - AUC)$, del cual el área bajo la curva Kendall es sólo una normalización con un factor de $\frac{2}{n^2}$. Esta relación entre la distancia Kendall τ y el AUCC no es nueva, sin embargo, las curvas Kendall lo exponen de forma más explícita [19].

CAPÍTULO 3
Implementación de la librería *CostCurves* para evaluar las prestaciones de un clasificador

"De alguna manera, R cambió mi opinión sobre el género humano al observar cuántas personas están realmente dispuestas a participar en actividades colectivas, buscando algo que trasciende a sus propios intereses. En este ámbito, se realizan muchas actividades sin que haya un reconocimiento individual."

Ross Ihaka, cofundador de R

La creación de paquetes o librerías con funcionalidades específicas dentro de un determinado campo, es una actividad común en la comunidad científica, sobre todo porque contribuye a la reutilización y mejora permanente de código, favorece la difusión de las líneas de investigación, facilita la experimentación y provee una forma elegante de compartir el trabajo investigado ya que exige un esfuerzo adicional para pulir las funciones y elaborar la documentación.

3.1 Lenguaje de programación R

La implementación de la librería `CostCurves` para evaluar las prestaciones de un clasificador, se ha llevado a cabo utilizando el entorno de programación R. Una de las motivaciones para usar este software es que proporciona un lenguaje especializado en el análisis de datos, posee una amplia variedad de métodos estadísticos y ofrece funcionalidades gráficas muy potentes. Estas características le han convertido en el lenguaje preferido para investigación y estadística. Surgió como parte del proyecto GNU de S desarrollado por Bell Laboratories por John Chambers et al. [31], por lo tanto es un software de acceso libre, gratuito y abierto.

R es un lenguaje multiparadigma, que soporta paradigmas como: orientación a objetos, imperativo, procedural, orientado, vectorial y funcional imperativo. Se destaca el hecho de que R es:

- Es altamente extensible, lo que significa que los usuarios pueden incorporar nuevas funciones.

- Tiene una comunidad muy dinámica, integrada por estadísticos de gran renombre. Lo que facilita el aumento del número de paquetes publicados en diversas áreas del conocimiento.
- Permite combinar análisis pre-empaquetados con análisis ad-hoc, específicos para una situación particular.
- Tiene extensiones específicas para áreas como bioinformática, minería de datos, geo-estadística, aprendizaje automático, entre otros.
- Proporciona un lenguaje de programación que puede integrarse con C, C++ y Fortran, lo que permite mejorar la eficiencia y rendimiento de las funciones.
- Se puede integrar con diversas bases de datos y librerías de otros lenguajes de programación interpretados como Python o Perl.
- Se distribuye bajo la licencia GNU GLP y está disponible para los sistemas operativos: Windows, Macintosh, Unix y GNU/Linux.

La mayor desventaja que presenta este lenguaje es que la curva de aprendizaje es costosa y la interacción con el usuario es poco amigable.

En cuanto a la implementación de una librería en este lenguaje, R exige la verificación de altos estándares, respecto a la escritura del código y la documentación. Esto conlleva por un lado al uso de buenas prácticas de programación, con el fin de definir funciones de alta calidad y con una documentación adecuada para su reutilización y distribución. Sin embargo, todo este proceso puede resultar complicado y tedioso.

3.2 Metodología

Una librería, paquete o *package* en inglés no es más que una colección de objetos creados y organizados siguiendo un protocolo que garantiza un soporte mínimo para el usuario y la ausencia de errores sintácticos en la programación.

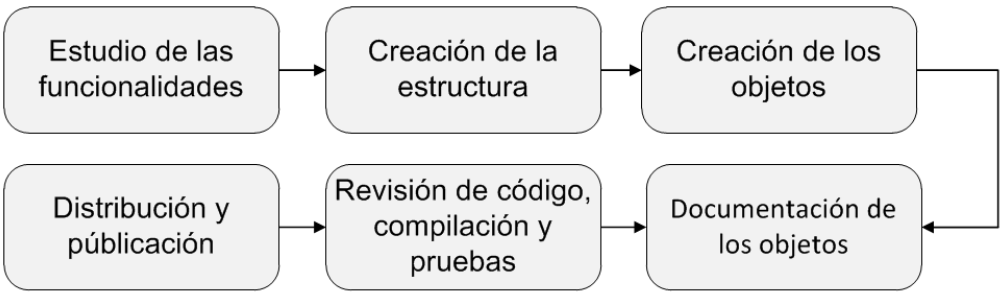


Figura 3.1: Metodología de implementación de una librería en R

R proporciona una guía muy completa en la que se detalla los requisitos y pasos necesarios para la creación de nuevos paquetes. En base a estas recomendaciones se describe la metodología utilizada para la construcción de este paquete, que consta de seis etapas, y que se representa por el esquema de la Figura 3.1.

3.2.1. Estudio de las funcionalidades

Antes de empezar a programar es importante realizar un estudio previo de las funcionalidades que se desean plasmar en la librería y además se debe investigar si existen funciones similares en otros paquetes. Esta exploración previa ayuda a reutilizar código y evita la publicación de librerías con prestaciones duplicadas.

3.2.2. Creación de la estructura

La segunda fase de la metodología comprende la definición y creación de la estructura o esqueleto del paquete [25]. El esqueleto del paquete es la estructura de ficheros y archivos necesarios para que R pueda compilar la librería. La estructura básica de un paquete se puede observar en la Figura 3.2.

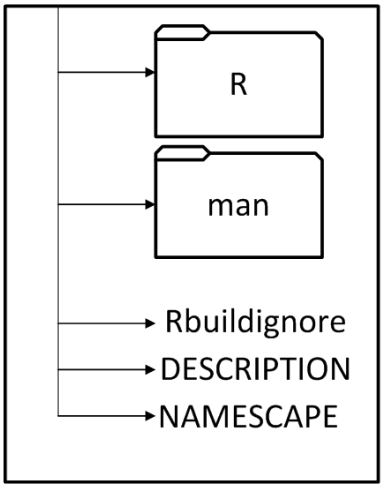


Figura 3.2: Estructura típica de una librería en R

En el nivel superior están los ficheros que definen la librería y que brindan información, como:

- `Rbuildignore`: Proporciona indicaciones para la creación del paquete.
- `DESCRIPTION`: Es un fichero de texto plano, fundamental en la librería, contiene la descripción del paquete y sus características principales. R exige un mínimo de campos que se deben cumplimentar obligatoriamente, como: título, descripción, autores, tipo de licencia otorgada al paquete, entre otros.
- `NAMESPACE`: Es necesario cuando se utilizan funciones de paquetes externos a la librería. Es en este fichero donde se deben declarar las librerías adicionales que se están utilizando en el paquete.

Por otra parte también están los subdirectorios en donde se almacenarán los objetos y la documentación de la librería, existen dos subcarpetas obligatorias que son:

- `R`: subcarpeta que contiene todo los objetos del tipo función.

- `Man`: subcarpeta en la que se encuentran los ficheros `.Rda`, en los que se escribe la documentación de ayuda para el paquete.

Opcionalmente se pueden añadir otros directorios para agregar otros objetos o documentación adicional como por ejemplo:

- `Src`: código en un lenguaje de bajo nivel (opcional)
- `Data`: bases de datos (opcional)

3.2.3. Creación de objetos

Una vez que se ha creado la estructura de la librería, la siguiente fase es la creación de los objetos que contendrá el paquete, estos pueden ser de cualquier tipo pero los más utilizados son del tipo función o datos. En esta etapa juega un papel muy importante la habilidad y el nivel de experticia del programador, ya que se requiere no solo que el código esté libre de errores en la sintaxis, sino que, además sea claro y fácil de interpretar. Una manera de conseguirlo es haciendo uso de las buenas prácticas para el desarrollo de software, algunas recomendaciones tomadas del manual de «Buenas Prácticas en Ciencias de Computación» [43] se citan a continuación:

- Los programas deben escribirse para personas y no para computadores.
- Se deben automatizar tareas repetitivas.
- Se deben realizar cambios incrementales.
- Se debe hacer uso de control de versiones.
- Se debe elaborar un plan para depurar errores.
- La optimización del software se debe hacer sólo después de que funciona correctamente.

Funciones (`function`)

Una función en R es un conjunto de expresiones o instrucciones agrupadas bajo un determinado nombre, que realizan una tarea determinada utilizando los parámetros o argumentos que le sean asignados. Las asignaciones dentro de la función son temporales, es decir, que sólo se almacenan durante el tiempo de ejecución.

La creación de funciones permite añadir nuevas funcionalidades. Todas las funciones disponibles en R mantienen el mismo *status* y las que han sido escritas por otros usuarios se pueden reescribir para adaptarlas a una necesidad particular. Típicamente la definición de una función tiene la siguiente forma [23]:

```
1 nombre <- function(arg1, arg2, arg3,..., argn){instrucciones}
```

No hay ninguna regla específica para definir el nombre de la función, pero se debe evitar el uso de números o caracteres especiales y es preferible no usar nombres previamente designados a otras funciones.

Las instrucciones son un conjunto de expresiones que utilizan los argumentos para calcular un determinado valor. El valor que retorna la función se conoce como salida o respuesta.

Para llamar a una función se suele utilizar la siguiente sintaxis:

```
nombre(arg1, arg2, arg3,..., argn)
```

Una de las facilidades de R al momento de ejecutar una llamada a una función, es que los parámetros se pueden colocar en cualquier orden, por lo que es necesario especificar para cada entrada el nombre del argumento al que corresponde, y si estos se colocan en el orden con el que se han definido en la función, este paso se puede obviar. Otra opción es definir valores por defecto para uno o más argumentos, de este modo el usuario no estará obligado a incluirlo en la llamada de la función a menos que requiera cambiar su valor.

Datos (`data`)

Los datos son objetos de R, generalmente almacenados en ficheros con extensión `.Rda` (`Rdata`). El lenguaje de programación acepta múltiples tipos de datos y ofrece facilidades para transformarlos de un formato a otro. Sin embargo, es preferible no utilizar otro formato ya que este ocupa menos espacio y su ejecución es más rápida.

La mayoría de las librerías publicadas en R, suelen incluir datos debido a que proporcionan casos de uso probados para las funciones del paquete. Además, si la audiencia del paquete es mucho más específica, es una buena manera de distribuir los datos ya que al igual que las funciones exige la incorporación de documentación.

Hay tres formas principales para incluir los datos en un paquete (estrictamente ligados al uso y su audiencia):

- Si se desea almacenar datos binarios y que estén disponibles para el usuario, se deben guardar en el directorio data.
- Si se requiere almacenar datos analizados, pero que no estén disponibles al público, se hace uso del fichero `sysdata.rda`. Este es el mejor lugar para poner los datos que requieren las funciones.
- Si interesa almacenar los datos en bruto, estos se deben colocar en el directorio inst/extdata.

Una alternativa sencilla a estas tres opciones es incluirlo en la fuente del paquete. Esto se puede hacer de forma manual o utilizando un comando para serializar los datos incluidos en el código de R.

Para que los datos del paquete no ocupen memoria cuando no se estén utilizando es recomendable añadir en el fichero `DESCRIPTION`, la opción `Lazydata: TRUE`, de esta forma solo se cargarán los datos cuando se vayan a utilizar. Esto es especialmente importante cuando el tamaño del conjunto de datos es muy grande.

3.2.4. Documentación de objetos

La cuarta fase consiste en la redacción de la documentación de la librería, seguramente es una de las etapas más complicadas pero irrenunciables dentro de todo el proceso. La redacción de la documentación debe ser clara, completa y concisa; de preferencia se redactar en el idioma inglés.

Además, se debe proporcionar ejemplos de uso válidos, es decir, que cuando se ejecuten lo hagan sin errores de sintaxis. Esto por dos razones: la primera es porque R probará cada uno de los ejemplos y si encuentra errores en el paquete no los compilará y la segunda razón, y no menos importante, es porque estos ejemplos servirán de ayuda a futuros usuarios del paquete.

Los archivos de documentación de la librería se escriben en un lenguaje genérico de R, llamado Rd, cuya sintaxis es muy similar a LATEX. Para cada función de la carpeta R o para cada conjunto de datos de la carpeta data, se debe tener un fichero `.Rd`, estos ficheros se deben almacenar en el directorio `MAN`.

La principal ventaja de utilizar este lenguaje para documentar la librería es que: le permite a R trasladar los ficheros `.Rd` a documentos HTML, de texto o LATEX, según se requiera. Cada fichero `.Rd` está obligado a incluir determinadas secciones, pero además se puede añadir secciones opcionales.

En el capítulo 2 del WRE *(Writing R extensions)* [40], hay un extenso conjunto de instrucciones para dar formato a la documentación, este paso se debe manejar con especial cuidado, ya que la mayoría de los errores que se presentan en la compilación ocurren por problemas en la redacción de la documentación. Además, los mensajes de error proporcionados por R pueden ser bastante críticos, por lo que es de mucha utilidad seguir algunas recomendaciones generales dadas por Peter Rossi [34]:

- Crear el archivo Rd de forma incremental, esto supone no escribir un archivo completo de corrido. Se puede empezar con un archivo simple incluyendo solamente los campos obligatorios y una vez que funcione se pueden agregar instrucciones de formato u otras opcionales. Cada cambio se debe probar uno a uno.

- Trabajar con una única función cada vez, es decir, que se debe evitar esperar que el paquete esté escrito por completo incluyendo sus archivos Rd para empezar a depurarlo, ya que podría resultar una tarea muy tediosa y complicada. Es mejor ir depurando una función y un archivo Rd a la vez.

- Revisar el código de todas las funciones y ejecutar cada uno de los ejemplos dados.

El archivo con extensión `.Rd` para una función debe contener como mínimo los siguientes puntos: nombre, alias, título, descripción y modo de empleo. Adicionalmente se pueden añadir otras secciones como: argumentos, autor, ejemplo, etc. En promedio un fichero `.Rd` consta de diez apartados, cada uno de los cuales empieza por su correspondiente instrucción.

La documentación de un objeto tipo datos es muy similar al de una función con contadas diferencias, en este caso es necesario agregar dos etiquetas: formato y fuente. La primera se utiliza para especificar el tipo de datos y la segunda para indicar la fuente de la que se ha extraído la información.

3.2.5. Revisión de código, compilación y pruebas

Si se han tomado en cuenta las recomendaciones mencionadas en la fase anterior, el proceso de revisión de código resultará muy sencillo. Sólo hará falta ejecutar un par de comandos y R se encargará de crear los archivos de documentación en los formatos de texto, LaTeX y HTML, compilará el código fuente, chequeará las inconsistencias y varios errores, ejecutará los ejemplos y finalmente construirá un manual en formato DVI. Adicionalmente, R mostrará un informe detallado de la revisión, en este punto es importante verificar que el tiempo de ejecución de los ejemplos no supere los dos minutos. También se debe chequear que los mensajes de advertencia no sean relevantes, esto significa que aunque R indique alguna alerta, no se impida la compilación del paquete ni su ejecución posterior. En el caso de que existan errores graves se interrumpirá la ejecución y será necesario corregirlos y realizar un nuevo chequeo [41].

Una vez superado el proceso de revisión, el siguiente paso será la construcción del paquete, esto se consigue utilizando el comando build. Este comando se encarga de compilar el paquete y cargarlo al directorio de trabajo para que se pueda probar.

3.2.6. Distribución y publicación

La distribución de los paquetes en R se puede realizar de varias maneras, la más básica es utilizando `Source packages` o `Binary packages`. Ambos son ficheros comprimidos que se crean directamente en R, el primero requiere que los usuarios construyan el proyecto a través de una fuente en su estación de trabajo, normalmente se utiliza para instalar el paquete en Windows, el segundo se usa mayormente para instalar la librería en Linux, también pueden usarse en Windows o MAC pero requiere construcciones binarias.

Otra manera de distribuir un paquete es publicarlo en un repositorio de software como GitHub o uno más específico como R-Forge. Si el objetivo es compartirlo con una comunidad más amplia, se puede optar por realizar una contribución al CRAN *(Comprehensive R Archive Network)*, una colección de sitios web con el mismo contenido, que alojan extensiones de R, paquetes aportados por los usuarios y documentación.

Para cualquiera de las formas de distribución que se elija es necesario especificar el tipo de licencia asociada a la librería, en el archivo `DESCRIPTION`. El tipo de

licencia más utilizado es GNU-GPL (*GNU General Public license*) en sus diferentes versiones, aunque se pueden utilizar otros tipos de licencia que pueden poner más o menos restricciones en cuanto al uso de la librería.

3.3 Descripción y desarrollo de la librería

Siguiendo la metodología descrita en la sección anterior, se llevó a cabo una fase exploratoria con el fin de conocer si existían paquetes que ofrecieran funcionalidades similares o relacionadas con el tema. Se encontraron dos librerías publicadas en el CRAN que abordaban la temática: pROC [33] y ROCR [37]. Ambas dan un amplio soporte para el análisis ROC e incluyen funciones gráficas y analíticas muy robustas. Aunque ninguna de ellas incorpora funcionalidades sobre las Curvas de Coste, sirvieron de referencia en la nomenclatura empleada en variables, argumentos y salidas.

Dado que el propósito general de la librería es servir como herramienta de apoyo en las investigaciones referentes a la temática de las Curvas de Coste y los métodos de selección de umbral, el nombre asignado al paquete fue `CostCurves`. La base teórica que soporta las funciones implementadas, hace referencia a las últimas publicaciones científicas realizadas en esta área del conocimiento.

La librería sigue una estructura básica, con los archivos descriptivos requeridos: `DESCRIPTION` y `NAMESPACE` en el nivel superior. Además cuenta con tres subdirectorios que albergan los ficheros de los objetos y de la documentación. Tal y como se muestra en la Figura 3.3.

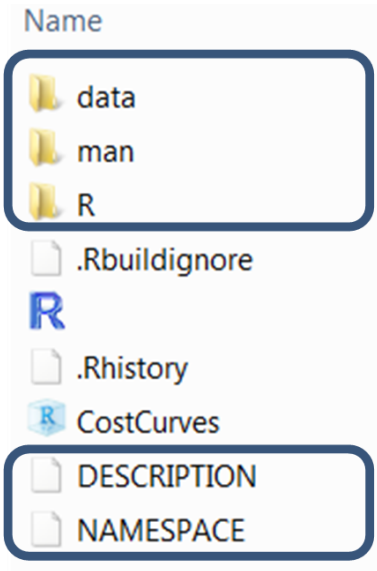


Figura 3.3: Estructura de la librería `CostCurves`

La librería `CostCurves` está integrada, en su mayoría, por objetos de tipo función (Figura 3.4). Sin embargo se ha agregado un objeto de tipo datos, con la finalidad de apoyar los ejemplos de uso de la librería. Básicamente se trata de un dataframe que consta de cinco columnas y diez filas. La primera columna tiene valores binarios que definen la clase de cada instancia, las siguientes cuatro co-

lumnas representan las predicciones de cuatro clasificadores diferentes A, B, C y D. Tanto los ejemplos como los datos fueron tomados de la referencia [9].

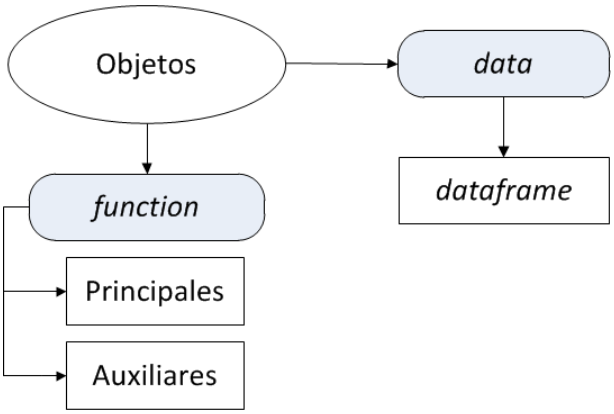


Figura 3.4: Tipos de objetos de la librería `CostCurves`

La librería también cuenta con una documentación muy completa (Apéndice 1), que le permitió pasar la revisión de R y su posterior compilación. Finalmente el paquete se ha publicado bajo la licencia GNU-GPL versión 3, en el repositorio GitHub y está disponible al público en la siguiente url: https://github.com/paumoal/CostCurves-R_package/

3.4 Descripción de las funciones de la librería

El paquete está dotado de un total de nueve funciones, ocho de las cuales ofrecen resultados gráficos (Figura 3.5). El core del paquete se compone de siete funciones gráficas, todas relacionadas con la temática de las Curvas de Coste.

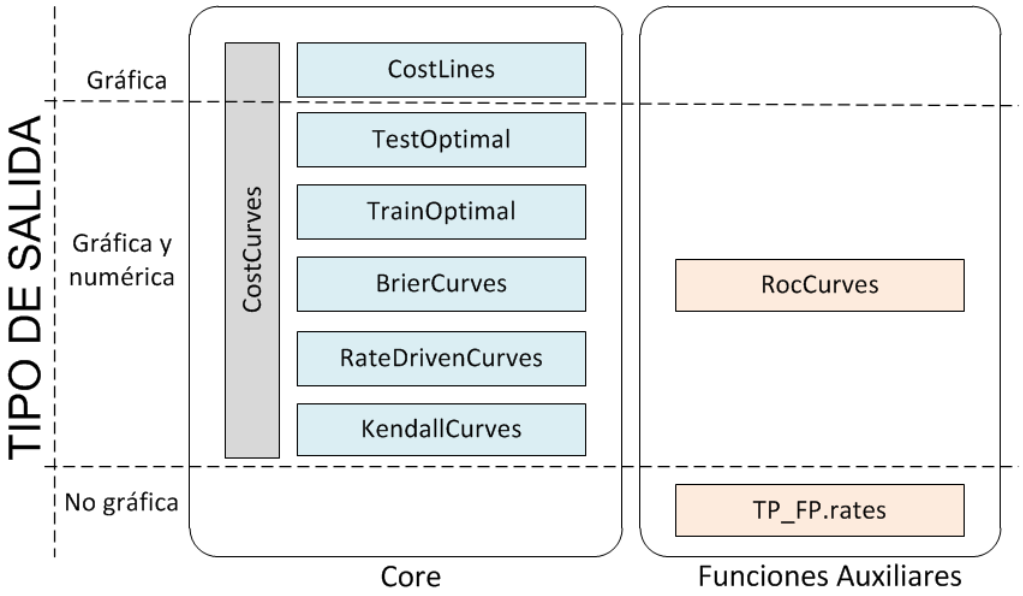


Figura 3.5: Pila de funciones del paquete `CostCurves`

La función CostLines grafica las líneas de Coste que corresponden a cada uno de los valores de umbral posibles. Las otras funciones que se muestran de co-

lor azul en el diagrama de la Figura 3.5, realizan el trazado de las Curvas de Coste utilizando los diferentes métodos de selección de umbral, estudiados en el capítulo 2. El nombre de cada función corresponde al nombre del método que implementan.

La función `CostCurves` ofrece la posibilidad de graficar todas las opciones anteriores utilizando una sola instrucción. De forma complementaria se ha incluido la función `RocCurves`, que dibuja las curvas ROC y su envolvente convexa.

Aunque la función `TP_FP.rates` no devuelve un resultado gráfico, es una función auxiliar que sirve de base en la construcción de las demás funciones, su objetivo es calcular la tasa de verdaderos positivos y la tasa de falsos positivos, dos ratios imprescindibles en el trazado de las Curvas de Coste y las curvas ROC.

3.4.1. Estructura del código de las funciones gráficas

La estructura del código de las funciones gráficas se compone de cuatro bloques (Figura 3.6), cada uno con un propósito específico.

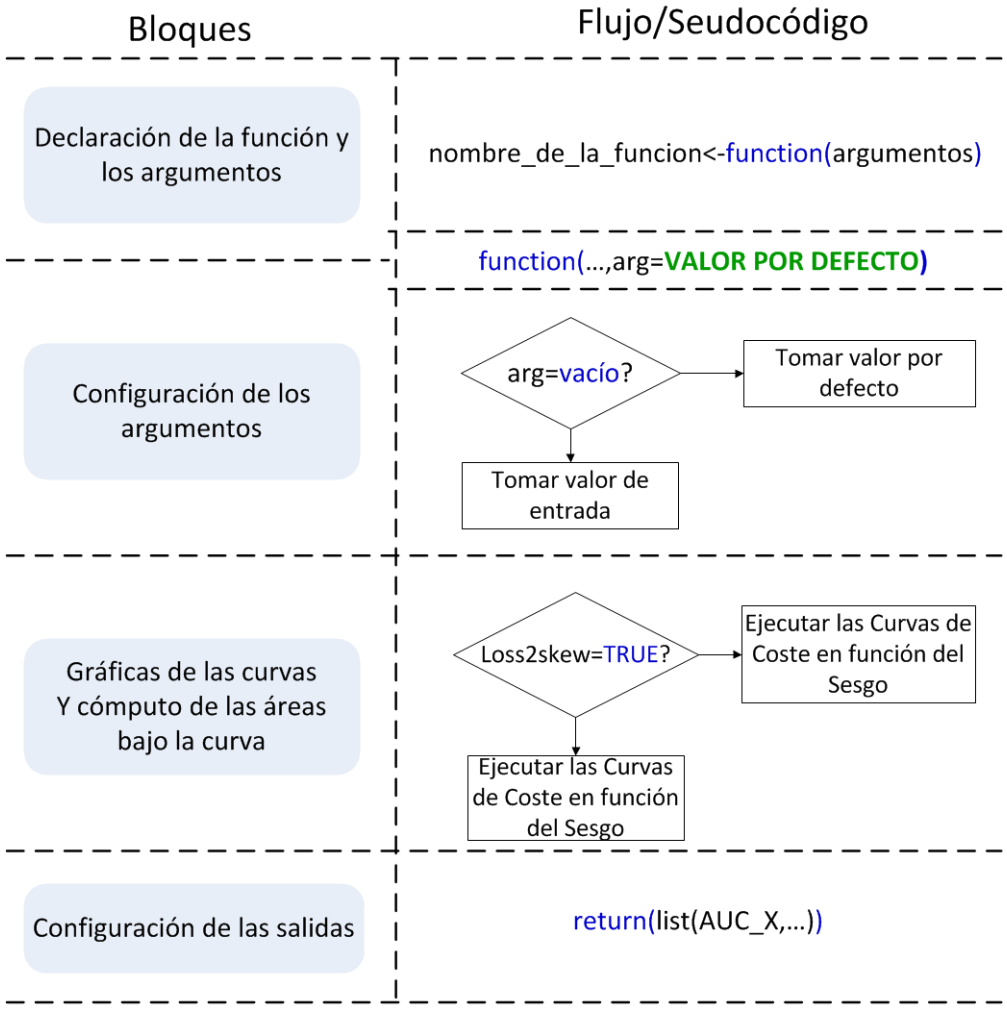


Figura 3.6: Tareas de cada bloque de la estructura del código de las funciones

- El primer bloque consiste en la declaración de la función y los argumentos, es decir, se da nombre a la función y se enuncian todos los posibles pará-

metros de entrada. Algunas entradas son opcionales, esto significa que no requieren que el usuario establezcan un valor.

- El segundo bloque está encargado de configurar y establecer todas las variables y/o funciones que se requieren para el trazado de las curvas y el cómputo de sus respectivas áreas bajo la curva. Esta configuración se puede realizar cuando se declaran los argumentos, fijando un valor por defecto (primer bloque) o después de la declaración de la función, a través del uso de condicionales que verifican si un argumento está vacío o no. En caso afirmativo se le asigna un determinado valor. La mayoría de las entradas opcionales se utilizan como opciones para personalizar las gráficas.
- En el tercer bloque se debe decidir qué código ejecutar para trazar las curvas (en función del coste o en función del sesgo) y calcular sus AUCCs. Esta decisión dependerá del valor del argumento `loss2skew`.
- El cuarto y último bloque es común a todas las funciones excepto `LinesCost`, ya que no ofrece ninguna salida numérica. En esta sección se forma una lista con los resultados de las diferentes opciones seleccionadas.

3.4.2. Entradas o argumentos

Cada función cuenta con argumentos obligatorios y opcionales, la mayoría son genéricos a casi todas las funciones. Los parámetros generales se describen en la Tabla 3.1.

Parámetro	Tipo	Descripción
predictions (obligatorio)	Lista de vectores	Probabilidades o scores asignados por el clasificador a un conjunto de datos
classes (obligatorio)	Lista de vectores	Clases asignadas por el clasificador a un conjunto de datos
loss2skew (opcional)	Booleano (TRUE o FALSE)	Selección del tipo de curva de coste a graficar: en función del coste o en función del sesgo.
uniquec (opcional)	Booleano (TRUE o FALSE)	El valor verdadero indica que se debe utilizar el mismo vector de clases para cada uno de los vectores de probabilidades o scores.

Tabla 3.1: Descripción de los argumentos generales de las funciones gráficas

La función `TrainOptimal` tiene parámetros adicionales (Tabla 3.2), debido a que por definición, para que pueda ser trazada exige el conjunto de predicciones y de clases asignadas al *dataset* de prueba, así como el conjunto de predicciones y de clases asignadas al conjunto de entrenamiento.

Por otra parte, ya que la función `CostCurves` permite el trazado de más de un tipo de Curva de Coste, incorpora argumentos para activar los diferentes métodos de selección de umbral. (Tabla 3.3)

Parámetro	Tipo	Descripción
predictions_train (obligatorio)	Lista de vectores	Probabilidades o scores asignados por el clasificador a un conjunto de datos de entrenamiento
classes_train (obligatorio)	Lista de vectores	Clases asignadas por el clasificador a un conjunto de datos de entrenamiento
predictions_test (obligatorio)	Lista de vectores	Probabilidades o scores asignados por el clasificador a un conjunto de datos de prueba
classes_test (obligatorio)	Lista de vectores	Clases asignadas por el clasificador a un conjunto de datos de prueba
uniquecT (opcional)	Booleano (TRUE o FALSE)	Respecto al *dataset* de entrenamiento: el valor verdadero indica que se debe usar el mismo vector de clases, para cada uno de los vectores de probabilidades o scores asignados por el clasificador.
refuseT (opcional)	Booleano (TRUE o FALSE)	El valor verdadero indica que se debe usar el mismo vector de clases y de probabilidades o scores asignados al *dataset* de entrenamiento, para cada par de vectores (clases y probabilidades o scores) asignados a los *datasets* de prueba.

Tabla 3.2: Descripción de los argumentos de la función `TrainOptimal`

Parámetro	Tipo	Descripción
cost_lines	Booleano (TRUE o FALSE)	Activa o desactiva el trazado de las Líneas de Coste
test_optimal	Booleano (TRUE o FALSE)	Activa o desactiva el trazado de las Curvas de Coste seleccionando el umbral *Test optimal*
train_optimal	Booleano (TRUE o FALSE)	Activa o desactiva el trazado de las Curvas de Coste seleccionando el umbral *Train optimal* (Requiere la predicciones y clases del *train* y del *test*)
predictionsT	Lista de vectores	Probabilidades o scores asignados por el clasificador a un conjunto de datos de entrenamiento
classesT	Lista de vectores	Clases asignadas por el clasificador a un conjunto de datos de entrenamiento
uniquecT	Booleano (TRUE o FALSE)	Respecto al *dataset* de entrenamiento: El valor verdadero indica que se debe usar el mismo vector de clases, para cada uno de los vectores de probabilidades o scores asignados por el clasificador.
uniquec	Booleano (TRUE o FALSE)	Respecto al *dataset* de prueba: El valor verdadero indica que se debe usar el mismo vector de clases, para cada uno de los vectores de probabilidades o scores.
uniqueTrain	Booleano (TRUE o FALSE)	El valor verdadero indica que se debe usar el mismo vector de clases y de probabilidades o scores asignados al *dataset* de entrenamiento, para cada par de vectores (clases y probabilidades o scores) asignados a los *datasets* de prueba.
score_driven	Booleano (TRUE o FALSE)	Activa o desactiva la gráfica de las Curvas de Coste seleccionando el umbral *Score driven*
rate_driven	Booleano (TRUE o FALSE)	Activa o desactiva la gráfica de las Curvas de Coste seleccionando el umbral *Rate driven*
kendall_cuves	Booleano (TRUE o FALSE)	Activa o desactiva la gráfica de las curvas Kendall

Tabla 3.3: Descripción de los argumentos de la función `CostCurves`

3.4.3. Salidas

Todas las funciones gráficas, excepto CostLines, devuelven valores numéricos de tipo float que corresponden al cálculo del área bajo la curva (AUCC) de cada una de las Curvas de Coste trazadas. La Tabla 3.4 muestra la nomenclatura utilizada para las respectivas salidas de las diferentes funciones.

Función	Salidas	Tipo	Descripción
CostLines	NA	NA	Sin respuesta numérica, únicamente gráfica.
TestOptimal	break_points treshold AUCC_tO	Lista de vectores	break_points: Puntos de inflexión de la curva treshold: Valores de umbral para cada tramo de la curva. AUCC_tO: Área bajo la curva de la menor envolvente
TrainOptimal	AUC_TO	número decimal	Área bajo la Curva de Coste cuyo método de selección de umbral es el Train optimal
BrierCurves	AUBC	número decimal	Área bajo la Curva Brier
RateDrivenCurves	AURDC	número decimal	Área bajo la Curva Rate driven
KendallCurves	AURDC	número decimal	Área bajo la Curva Kendall
CostCurves	AUCC_tO AUCC_TO AUBC AURDC AUKC	Lista	Devuelve las áreas bajo la curva de las diferentes Curvas de Coste seleccionadas
RocCurves	AUC AUCH	Lista	AUC: Área bajo de Curva ROC AUCH: Área bajo la envolvente convexa
TP_FP.rates	TP FP	Matriz	TP: Primera columna de la matriz, representa la tasa de Verdaderos Positivos para los diferentes valores de umbral FP: Segunda columna de la matriz, representa la tasa de Falsos Positivos para los diferentes valores de umbral

Tabla 3.4: Descripción de la salida de las funciones del paquete `CostCurves`

El Apéndice 1 es el manual de uso del paquete `CostCurves`. Este provee mayor información acerca de la descripción, sintaxis, modo de uso y ejemplos de cada una de las funciones que conforman la librería. Información que podrá ser consultada en la sección de ayuda de R (`help`), después de instalar e importar el paquete.

3.4.4. Opciones gráficas

Una virtud de la librería es que ofrece la posibilidad de personalizar el aspecto de las gráficas. Esto se consigue utilizando únicamente los parámetros de entrada de las funciones. La mayoría de los nombres de los argumentos son iguales a los de cualquier otra función gráfica en R, con el fin de que la terminología sea familiar para el usuario.

Las opciones gráficas se han configurado para que sean argumentos opcionales de la función, esto implica que tengan asociados valores por defecto que se pueden cambiar siempre que se requiera. La Tabla 3.5, detalla brevemente cada una de las opciones gráficas disponibles en las funciones.

Parámetro	Tipo	Descripción
hold	Booleano (TRUE o FALSE)	Mantiene la vista del gráfico actual, para permitir que se grafiquen nuevos trazos por encima.
plotOFF	Booleano (TRUE o FALSE)	Activa o desactiva el trazo de las curvas.
gridOFF	Booleano (TRUE o FALSE)	Activa o desactiva la grilla.
pointsOFF	Booleano (TRUE o FALSE)	Activa o desactiva el trazo de las curvas.
legendOFF	Booleano (TRUE o FALSE)	Activa o desactiva la impresión de la legenda.
main	Cadena de caracteres	Título de la gráfica.
ylab	Cadena de caracteres	Etiqueta del eje y.
xlab	Cadena de caracteres	Etiqueta del eje x.
namesClassifiers	Vector de cadenas	Establece los nombres de los clasificadores que se están graficando.
lwd	Vector de valores enteros	Define el ancho de la línea.
lty	Vector de valores enteros o caracteres	Define el tipo línea.
pch	Vector de valores enteros o caracteres	Define el ancho del punto.
cex	Vector de valores enteros	Define el tipo del punto.
col	Lista de vectores o Vector, con valores enteros, caracteres o hexadecimal	Define los colores de cada curva.
xPosLegend	Número decimal entre 0 o 1	Coordenada x para la posición de la leyenda.
yPosLegend	Número decimal entre 0 o 1	Coordenada y para la posición de la leyenda.
cexL	Número decimal	Ajusta el tamaño de la leyenda.

Tabla 3.5: Opciones gráficas de la librería `CostCurves`

CAPÍTULO 4
Casos de uso

"El auténtico genio consiste en la capacidad para evaluar información incierta, aleatoria y contradictoria."

Winston Churchill, estadista

En este capítulo se presentan dos casos de uso de la librería. El primer caso se enfoca en el análisis y visualización del rendimiento de un clasificador en etapa de desarrollo. El segundo caso muestra la comparación de las prestaciones de dos clasificadores utilizando las Curvas de Coste y los diferentes métodos de selección de umbral.

Previo al análisis y uso de la herramienta para la evaluación del clasificador, es necesario revisar grosso modo el proceso típico de desarrollo y evaluación de un modelo de clasificación (Figura 4.1).

Una fase muy importante de este proceso está relacionada con la extracción y tratamiento de los datos. Debido a que el clasificador aprende a través de un *dataset* de entrenamiento, gran parte del tiempo de desarrollo se invierte en preparar los datos antes de suministrarlos al modelo. En esta etapa, también se realiza la partición de los datos, que consiste en dividir el conjunto de datos en dos subconjuntos: uno para el entrenamiento y otro para la evaluación. Existen varias técnicas para seleccionar el *dataset* de entrenamiento y el *dataset* de prueba, en general todas ellas buscan tomar un subconjunto que sea lo más representativo posible.

Dado a que el propósito de los casos de uso es mostrar la utilidad del paquete implementado en la evaluación de modelos de clasificación, se presta menos atención al preprocesamiento de los datos y se asume que los *datasets* (de entrenamiento y de prueba) han sido preparados adecuadamente y servirán para generar un modelo aceptable.

Para extraer los datos e implementar los modelos se ha utilizado una librería dedicada al aprendizaje automático, llamada RWeka.

RWeka es una interfaz de R para utilizar Weka (Waikato Environment for Knowledge Analysis) [38]. Weka es una plataforma de software libre desarrollada en JAVA, que ofrece múltiples funcionalidades de aprendizaje automático. Contiene herramientas para el preprocesamiento de datos y una serie de algoritmos de clasificación, regresión, agrupamiento y reglas de asociación [16].

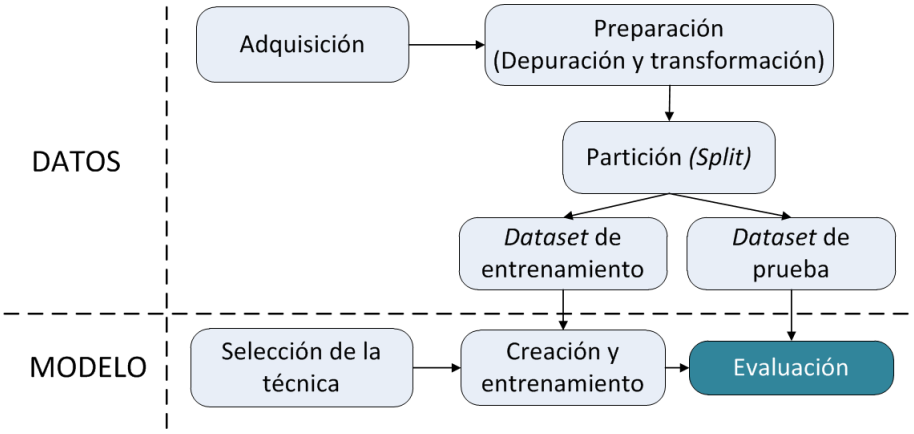


Figura 4.1: Proceso de creación de un modelo de clasificación

El conjunto de datos que se va a utilizar en los dos casos de uso, corresponde a un *dataset* con información financiera. Su nombre es `credit-a` y fue tomado del repositorio de la UCI (Universidad de California Irving), un repositorio de *datasets* especializado en *machine learning* [2].

En cuanto a los modelos de clasificación que van a ser evaluados, se han empleado dos técnicas: J48 y Regresión Logística. Sus características más relevantes se muestran en la Tabla 4.1.

Clasificador	Características
J48	→ Implementación del algoritmo C4.5 en Weka. → C4.5. es un árbol de decisión. → Algoritmo de inducción que genera una estructura de reglas o árbol a partir de subconjuntos (ventanas) de casos extraídos del conjunto total de datos de entrenamiento. → Se conoce como clasificador estadístico. → Usa conceptos de entropía de información.
Regresión Logística	→ Tipo de análisis de regresión. → Predice variables categóricas, en función de las variables independientes o predictoras. → Es un modelo GML (Modelos Lineales Generalizados) que usa como función de enlace la función logit. → Las probabilidades que describen el posible resultado de una única prueba se modelan, como una función de variables explicativas, utilizando una función logística. → Es conocido como modelo logístico, modelo logit o clasificador de máxima entropía.

Tabla 4.1: Características de clasificadores (J48 y Regresión logística) [18] [44]

El Apéndice 2, muestra el detalle de las sentencias de R utilizadas para crear los casos de uso y también ofrece otros ejemplos. Se debe mencionar que debido a que la configuración de los modelos depende de los datos de entrada, los mode-

los que se obtendrían si se siguiera paso a paso las instrucciones del Apéndice 2, podrían presentar diferencias respecto a los modelos que se muestran en este trabajo, ya que los subconjuntos de entrenamiento y de prueba se obtienen de forma aleatoria. Los cambios en los modelos también afectarían a las representaciones de las Curvas de Coste y los valores de sus respectivas AUCCs. Sin embargo, el análisis es aplicable independientemente de las variaciones en los resultados.

4.1 Caso 1: Desarrollo y evaluación de un clasificador

El procedimiento en R, empieza importando las librerías que contienen las funciones necesarias para tratar los datos y trabajar con los modelos. Esto se realiza utilizando la instrucción `library` seguido (entre paréntesis) del nombre de la librería a importar.

Las librerías requeridas en este caso son: RWeka, caret y por su puesto la nueva librería denominada `CostCurves`. Una vez que se tienen cargados los paquetes con sus respectivas funciones, el siguiente paso consiste en cargar los datos al espacio de trabajo. Esto se consigue con el comando `read.arff` tal y como se muestra en la Figura 4.2.

```
#Carga de Datos
data <- read.arff("credit-a.arff")
summary(data)
```

```
##  A1           A2               A3              A4          A5
##  b   :468   Min.   :13.75   Min.   : 0.000   u   :519   g   :519
##  a   :210   1st Qu.:22.60   1st Qu.: 1.000   y   :163   p   :163
##  NA's: 12   Median :28.46   Median : 2.750   l   :  2   gg  :  2
##             Mean   :31.57   Mean   : 4.759   t   :  0   NA's:  6
##             3rd Qu.:38.23   3rd Qu.: 7.207   NA's:  6
##             Max.   :80.25   Max.   :28.000
##             NA's   :12
##  A6          A7           A8          A9       A10
##  c      :137   v      :399   Min.   : 0.000   t:361   t:295
##  q      : 78   h      :138   1st Qu.: 0.165   f:329   f:395
##  w      : 64   bb     : 59   Median : 1.000
##  i      : 59   ff     : 57   Mean   : 2.223
##  aa     : 54   j      :  8   3rd Qu.: 2.625
##  (Other):289   (Other): 20   Max.   :28.500
##  NA's   :  9   NA's   :  9
##       A11        A12      A13          A14              A15          class
##  Min.   : 0.0   t:316   g:625   Min.   :   0   Min.   :     0.0   +:307
##  1st Qu.: 0.0   f:374   p:  8   1st Qu.:  75   1st Qu.:     0.0   -:383
##  Median : 0.0           s: 57   Median : 160   Median :     5.0
##  Mean   : 2.4                   Mean   : 184   Mean   :  1017.4
##  3rd Qu.: 3.0                   3rd Qu.: 276   3rd Qu.:   395.5
##  Max.   :67.0                   Max.   :2000   Max.   :100000.0
##                                 NA's   :13
```

Figura 4.2: Resumen del *dataset* `credit-a`

Realizando una exploración de los datos se observa que el *dataset* seleccionado, contiene 690 instancias cada una con 15 atributos de tres tipos diferentes (categóricos, reales y enteros), en la última columna se incluye la clase.

Este *dataset* contiene información relativa a los procesos de aplicación de tarjetas de crédito en los que se desea discriminar en función de una serie de características si una solicitud de crédito se debe aprobar (clase positiva) o rechazar (clase negativa). Por lo tanto, este conjunto de datos es ideal para crear modelos de clasificación binaria. Otro aspecto importante de este *dataset* es que tiene pocos valores faltantes y su distribución de clases no está balanceada.

Después de identificar los atributos de las instancias, la configuración de los datos y la variable a predecir, se realiza la partición de los datos para obtener dos subconjuntos, uno para el entrenamiento y otro para la evaluación. En este caso la partición de los objetos se realiza de forma aleatoria intentando que el número total de elementos de cada conjunto sea similar. Después de crear el *dataset* de entrenamiento se pasa a la siguiente etapa que consiste en la creación del modelo.

El modelo que se va a analizar en este caso es el árbol de decisión J48 y se implementa utilizando la instrucción: `J48(args)`. Los parámetros de entrada (args) que obligatoriamente se tienen que especificar son: la variable objetivo (variable a predecir) y el conjunto de datos de entrenamiento. La Figura 4.3 muestra un resumen con las principales características del modelo J48, como se puede observar se ha entrenado utilizando 330 ejemplos, de los cuales 311 se han clasificado correctamente, mientras que los 19 restantes no. El resumen también refleja la estimación del error absoluto y una serie de estadísticos que ofrecen pistas de la calidad del modelo, finalmente se proporciona la matriz de confusión con la que se podrá calcular otras medidas escalares de rendimiento como: el *accuracy*, el *recall*, el error positivo, la precisión, entre otros.

```
## === Summary ===
##
## Correctly Classified Instances         311               94.2424 %
## Incorrectly Classified Instances        19                5.7576 %
## Kappa statistic                          0.8837
## Mean absolute error                      0.1707
## Root mean squared error                  0.24
## Relative absolute error                 34.4323 %
## Root relative squared error             48.1908 %
## Total Number of Instances              330
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  139  11 |   a = +
##    8 172 |   b = -
```

Figura 4.3: Resumen de las características del modelo J48

La Figura 4.4, presenta un fragmento del árbol de decisión J48, básicamente indica el conjunto de reglas aprendidas por el modelo, que se usarán para clasificar una nueva instancia.

Como se puede notar los valores de pertenencia de una instancia a una clase (columna probstrain o probstest de la Figura 4.5(a) *train* y Figura 4.5(b) *test*) vie-

nen dados por probabilidades, esto implica que están acotados en el rango entre 0 y 1.

```
## J48 unpruned tree
## ------------------
##
## A9 = t
## |   A10 = t
## |   |   A3 <= 5.29
## |   |   |   A6 = c: + (10.0/1.0)
## |   |   |   A6 = d: + (2.0)
## |   |   |   A6 = cc: + (6.0)
## |   |   |   A6 = i: + (4.0/1.0)
## |   |   |   A6 = j: + (0.0)
## |   |   |   A6 = k
## |   |   |   |   A3 <= 0.585: + (2.0)
## |   |   |   |   A3 > 0.585: - (3.0/1.0)
## |   |   |   A6 = m: + (6.0)
## |   |   |   A6 = r: + (0.0)
## |   |   |   A6 = q
## |   |   |   |   A11 <= 2
                .
                .
                .
```

Figura 4.4: Fragmento del Árbol de decisión J48

```
print(predicciones_entrenamiento[muestra,])
```

```
##     cltrain probstrain
## 522       0 0.02702703
## 662       1 0.95833333
## 97        1 0.88888889
## 44        0 0.33333333
## 211       0 0.08333333
## 624       1 0.87500000
## 231       0 0.02702703
## 53        0 0.02702703
## 646       1 0.75000000
## 643       1 0.92307692
```

(a)

```
print(predicciones_prueba[muestra,])
```

```
##     cltest  probstest
## 232      0 0.01666667
## 662      1 0.92452830
## 93       1 0.80000000
## 179      0 0.01666667
## 376      1 0.88636364
## 52       0 0.75000000
## 216      0 0.80000000
## 534      1 0.60000000
## 459      1 0.77777778
## 236      0 0.01666667
```

(b)

Figura 4.5: Muestra de las predicciones del modelo (a) *train* y (b) *test*

Después de obtener las predicciones del modelo, se realiza la evaluación del rendimiento utilizando la librería `CostCurves`. La principal ventaja de tener una librería que empaquete todas las herramientas gráficas y las funcionalidades de las Curvas de Coste, es que se puede delegar la construcción de las curvas al software utilizando instrucciones simples y parametrizables.

En primer lugar, se contrastan las curvas ROC respecto a las Curvas de Coste *Test optimal*. Como se aprecia en la Figura 4.6(a) la curva ROC refleja el rendimiento del clasificador en función del *recall* y de la precisión, por lo que en general se podría considerar que el desempeño del clasificador J48 es aceptable. La curva ROC de color azul muestra el rendimiento del clasificador en la fase de entrenamiento, esta curva corresponde a la matriz de confusión de la Figura 4.3, es notorio que el modelo consiguió acertar las clases de las instancias con un porcentaje muy alto ya que fue entrenado con el mismo *dataset*. Sin embargo, cuando se proporciona el conjunto de datos de prueba (curva de color rojo), el número

de aciertos se reduce y por ende el clasificador consigue un menor rendimiento. El rendimiento óptimo en el *test* en el punto de inflexión de la curva de color rojo (aproximadamente en el par ordenado (0.18, 0.76)). Conociendo este valor se podría inferir las condiciones de operación óptimas del clasificador, aunque no se pueden identificar directamente a través del gráfico.

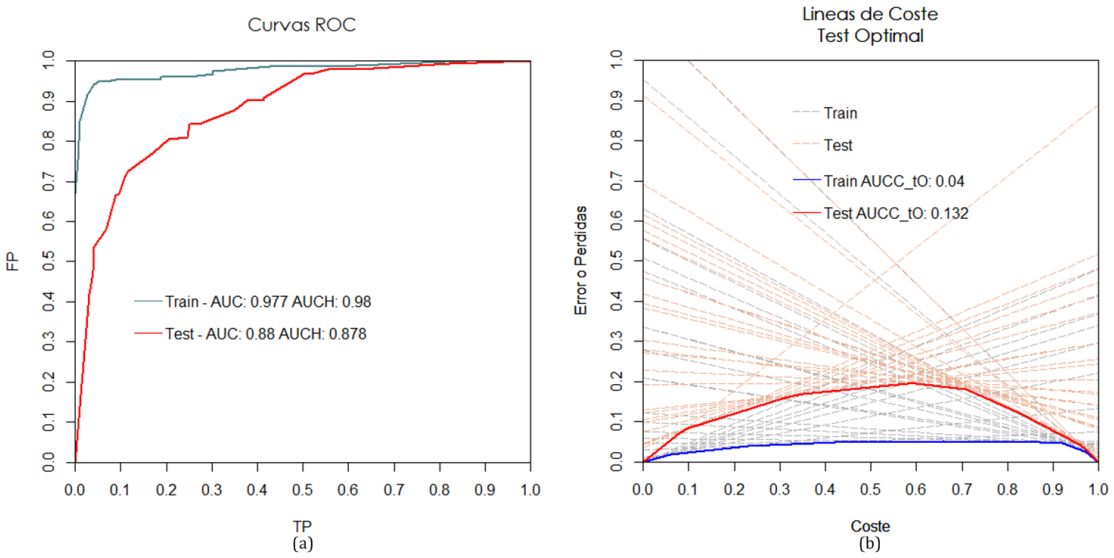


Figura 4.6: Curvas ROC vs. *Test optimal* (Evaluación del modelo J48)

Por otro lado, en la Figura 4.6(b) se muestran las líneas de coste y la Curva de Coste que resulta de la selección de umbral *Test optimal*. En este caso se observa que las pérdidas en el *test* aumentan considerablemente respecto al entrenamiento, puesto que el área bajo la curva *Test optimal* del *train* es menor que el AUCC del *Test optimal* del *test*. Las Curvas de Coste, permiten determinar por simple inspección las condiciones de operación que consiguen cada uno de los puntos óptimos.

Si se compara el método de selección de umbral *Train optimal* respecto a la Curva de Coste *Test optimal* (Figura 4.7), se observa que las condiciones óptimas del *test* están muy cercanos a los puntos de la curva que usa los valores óptimos de umbral del *train*, esto podría indicar que el modelo va a tener un comportamiento adecuado cuando pase a la fase de producción.

La figura 4.8, presenta la Curvas de Coste utilizando los métodos de selección de umbral *Score driven* y *Rate driven* respectivamente, estas curvas se comportan de manera muy similar a las dos anteriores, sin embargo el método de selección *Score driven* proporciona un rendimiento promedio ligeramente mayor que el resto de los métodos. Es necesario acotar que aunque a priori la medida del área bajo la curva nos da una pista de la efectividad del método de selección de umbral (mientras menor sea el AUCC, mejor será el rendimiento del clasificador), dependiendo de las condiciones de operación que se tengan se podrá determinar qué método ofrece mejores resultados, por ejemplo para las condiciones de operación dadas por el punto (0.9, 0.1), el método de selección de umbral *Rate driven* brinda un mayor rendimiento a pesar de que tiene el máximo valor de AUCC.

La Tabla 4.2 expone los diferentes valores de AUCC para cada uno de métodos de selección de umbral.

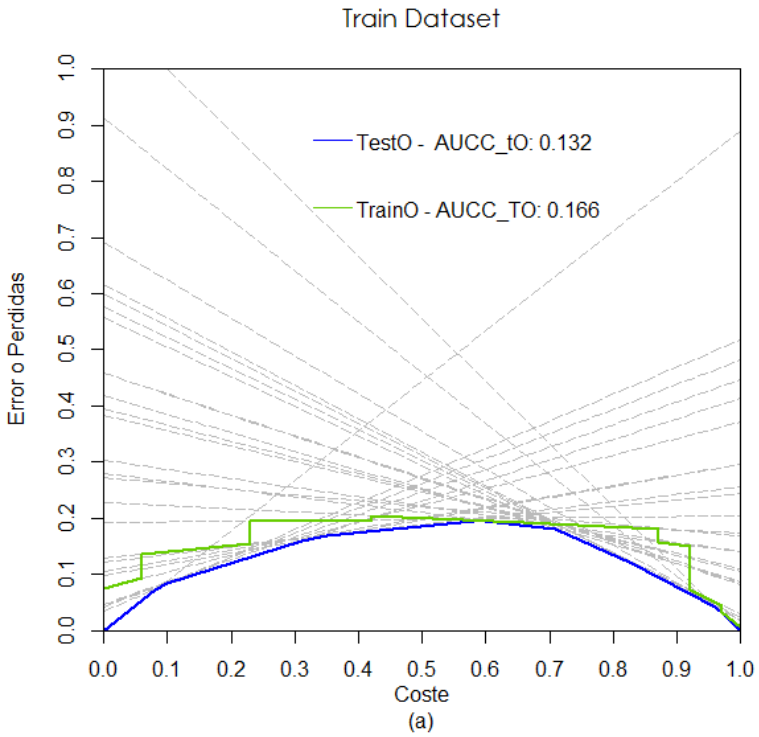


Figura 4.7: *Train optimal* del J48

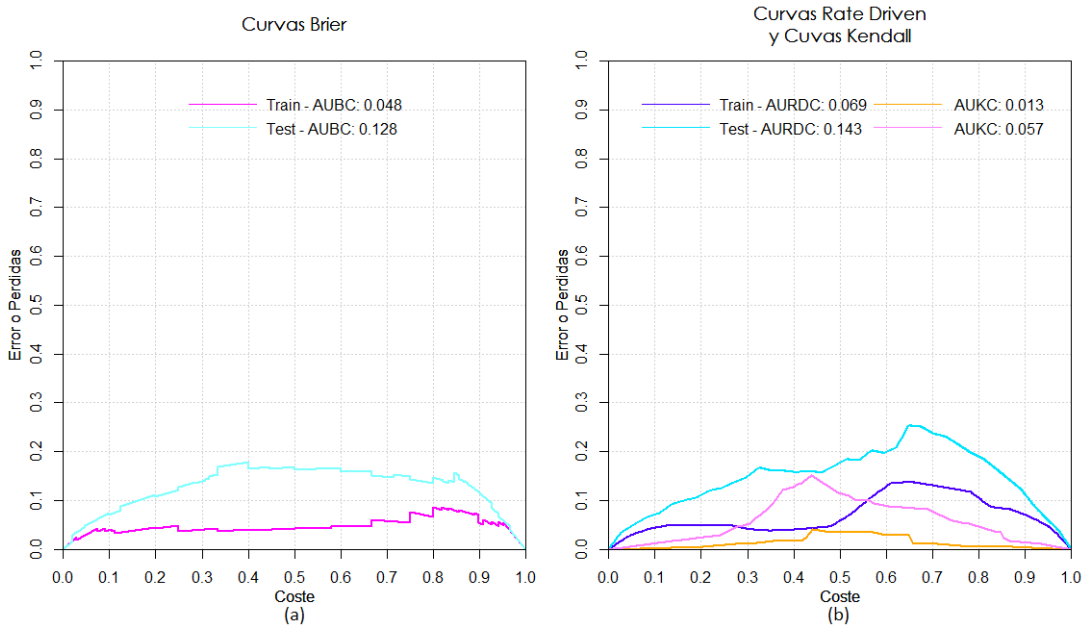


Figura 4.8: *Score driven*, *Rate driven*, y curvas Kendall del modelo J48

Métodos de selección de umbral	Área bajo la Curva	
	Train	Test
ROC	0.98	0.878
Test optimal	0.04	0.132
Train optimal	NA	0.166
Score driven	0.048	0.128
Rate Drive	.054	0.1430
Kendal	0.009	0.052

Tabla 4.2: AUC y AUCCs (*Train* y *Test* del J48)

En la Figura 4.9 se presentan las Curvas de Coste en función del sesgo utilizando todos los métodos de selección de umbral, se puede notar que si las clases del modelo fueran balanceadas, el rendimiento promedio del clasificador sería ligeramente mejor.

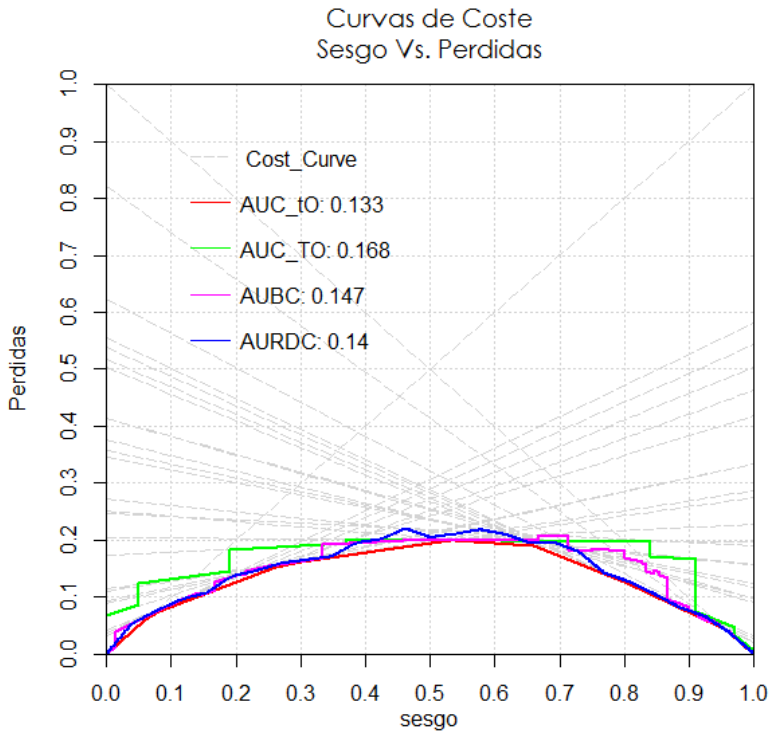


Figura 4.9: Curvas de Coste del J48
(32(a) en función del coste y 32(b) en función del sesgo)

Todas las gráficas presentadas en este estudio se obtuvieron utilizando las diferentes funciones que ofrece la librería `CostCurves`. Como se indica en el Apéndice 1, la apariencia de las gráficas se puede personalizar cambiando los valores por defecto de los parámetros gráficos de entrada. Por ejemplo si se desea superponer dos curvas únicamente será necesario establecer el parámetro `hold` con el valor `TRUE`.

4.2 Caso 2: Comparación del rendimiento de 2 clasificadores

En esta sección se realiza la comparación del rendimiento de dos clasificadores, para ello se construye un nuevo modelo de clasificación, usando la técnica de Regresión Logística. El *dataset* utilizado para crear el modelo es el mismo conjunto de datos del caso 1.

```
##
## === Summary ===
##
## Correctly Classified Instances         299               90.0602 %
## Incorrectly Classified Instances        33                9.9398 %
## Kappa statistic                          0.8006
## Mean absolute error                      0.1463
## Root mean squared error                  0.2681
## Relative absolute error                 29.6111 %
## Root relative squared error             53.9415 %
## Total Number of Instances
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  138  10 |   a = +
##   23 161 |   b = -
```

Figura 4.10: Resumen de las características del modelo de Regresión Logística

La instrucción que se utiliza para crear el modelo es: `Logistic(args)`, en la Figura 4.10 se muestra el resumen de sus características, como se puede observar el porcentaje de instancias clasificadas correctamente es menor respecto al del modelo J48, esto implica que usando los mismos datos de entrenamiento el modelo de Regresión Logística tiene un peor desempeño que el modelo J48. Sin embargo, cuando los modelos se evalúan con el *dataset* de prueba, se observa (Figura 4.11 y la Figura 4.12 (b, c y d)) que el rendimiento del modelo de regresión logística crece respecto al modelo J48.

Una medida escalar de rendimiento como la precisión o el *accuracy* solo proporciona una métrica general para conocer el desempeño de un clasificador, sin embargo, no permite visualizar el rendimiento del modelo para un determinado contexto. Por lo tanto, es imprescindible utilizar herramientas gráficas que nos aporten mayor información.

Una característica importante de la librería `CostCurves` es que permite graficar con una sola instrucción diferentes curvas de varios clasificadores. Para empezar el análisis se han graficado las curvas ROC de ambos clasificadores.

Si se observa la Figura 4.11, se distingue el punto de intersección de las dos curvas ROC, este cruce determina un umbral de decisión respecto al cual un clasificador será superior a otro. Aunque este valor de umbral permite determinar bajo qué condiciones se produce un mejor rendimiento, las curvas ROC en sí mismas no son capaces de responder a la pregunta de: qué clasificador ofrece mejores prestaciones para unas determinadas condiciones de operación, por lo que esta representación gráfica del rendimiento se queda muy limitada.

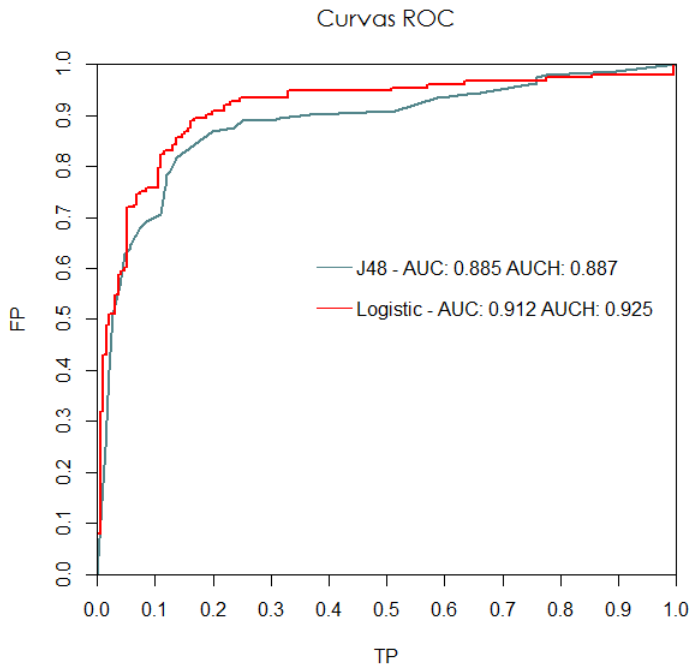


Figura 4.11: Curvas ROC (J48 vs. Regresión logística)

En contraparte las Curvas de Coste (Figura 4.12) permiten visualizar de forma directa el rendimiento en términos del error o pérdidas y del contexto.

En la Figura 4.12(a) se presentan las líneas de Coste y el *Test optimal* de cada modelo utilizando el *dataset* de prueba. Por su parte la Figura 4.12(b) muestra la curva *Train optimal* aplicando el conjunto de datos de prueba y el *Test optimal* usando el conjunto de datos de entrenamiento. Los métodos de selección de umbral restantes mostrados en las Figuras 34 (c, y d) se han graficado empleando el *testing dataset*.

	Área bajo la Curva	
Métodos de selección de umbral	**J48**	**Regresión Logística**
ROC	0.885	0.912
Test optimal	0.119	0.099
Train optimal	0.152	0.112
Score driven	0.128	0.097
Rate driven	0.143	0.074

Tabla 4.3: AUC y AUCCs (J48 vs. Regresión logística)

Las Curvas de Coste y los diferentes métodos de selección de umbral, reflejan que en general el rendimiento promedio del modelo de Regresión Logística sobresale al del clasificador J48 (Tabla 4.3). No obstante para algunas condiciones de operación el modelo J48 obtiene iguales o mejores resultados. Por ejemplo en la Figura 4.12(c) para algunos puntos del rango entre 0 y 0.33 ambos algoritmos consiguen el mismo resultado y para proporciones de coste mayores a 0.93 el J48

es claramente superior. En la Figura 4.12(d) también se puede distinguir el rango de condiciones de operación en el modelo de Regresión Logística obtiene menos pérdidas que el J48.

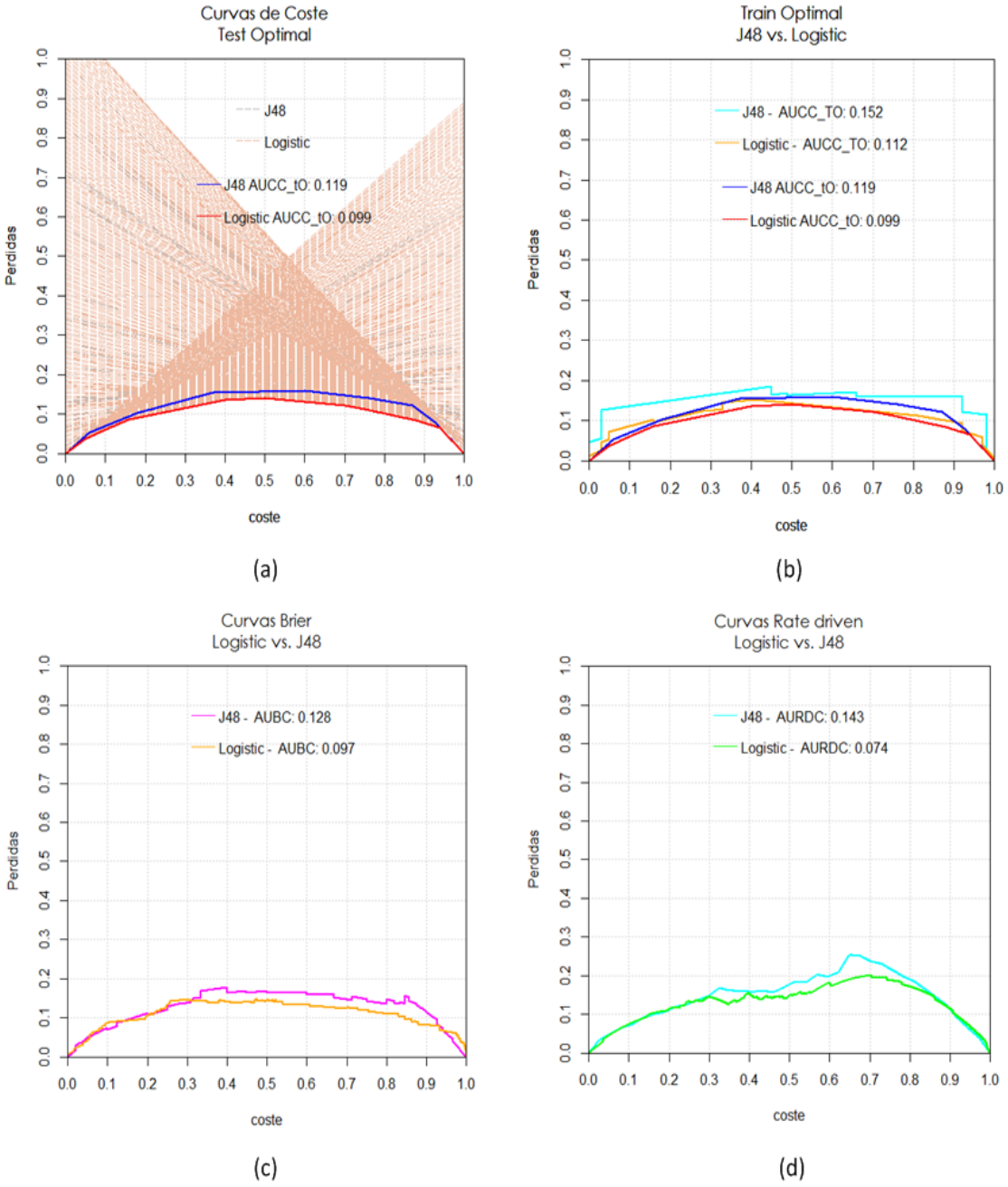


Figura 4.12: Curvas de Coste y métodos de selección de umbral (J48 vs. Regresión logística)

CAPÍTULO 5
Conclusiones

La construcción de paquetes o librerías que contienen funciones específicas para determinados campos es una tendencia cada vez más habitual en el ámbito académico y científico. Esto es debido a que facilitan el análisis de los problemas, permiten conseguir resultados con mayor rapidez y objetividad, y sirven de instrumento para que la interacción entre los sistemas de computación y los usuarios sea mucho más amigable. Así, una persona con mínimos conocimientos en informática, pero especialista en su área, puede desarrollar y probar modelos teóricos sin dedicar mucho tiempo a la programación.

R es sin duda uno de los lenguajes de programación más recomendables para el desarrollo de librerías robustas, especializadas en algún tema en concreto, ya que proporciona un entorno de programación muy completo y altamente escalable. A pesar de que la curva de aprendizaje de este lenguaje no es muy rápida en comparación con otros lenguajes de software libre o licenciado, R ofrece múltiples opciones para incorporar funcionalidades gráficas y estadísticas de forma relativamente sencilla. Además, posee una gran cantidad de documentación generada de forma propia o por su extensa comunidad de usuarios, esta última formada por profesionales de alto nivel de áreas muy variadas.

Por otra parte, los algoritmos de clasificación son cada vez más utilizados en aplicaciones prácticas de diversas áreas como: finanzas, negocios, medicina, marketing, entre otros. Por lo que se busca constantemente mejorar su desempeño.

La fase de evaluación cobra especial valor en el proceso de desarrollo de un clasificador, ya que de esta depende que el modelo obtenga resultados aceptables cuando pase a la fase de producción. Ergo, es fundamental definir métricas y métodos de evaluación efectivos que permitan determinar la fiabilidad del algoritmo y la capacidad para predecir correctamente variables que están ligadas a la toma de decisiones. Decisiones que en la mayoría de los casos son cruciales.

El análisis de las curvas de coste y los métodos de selección de umbral revisadas en este trabajo, contribuyen de manera significativa al campo del aprendizaje automático, ya que proveen una nueva concepción para visualizar y comparar el rendimiento de los clasificadores, tomando en cuenta dos parámetros muy frecuentes en aplicaciones de la vida real como son: las proporciones de clase y los costes asociados a errores en la clasificación.

La librería implementada en R, denominada `CostCurves`, además de ser un aporte nuevo al campo de la evaluación de los clasificadores, se ajusta al marco

teórico de las Curvas de Coste y los métodos de selección de umbral. Dicho ajuste ha podido ser validado por medio de pruebas y ensayos ejecutados a través de las funciones embebidas en el paquete `CostCurves` y empleando un conjunto de datos, tomado de una aplicación real, con modelos de clasificación altamente utilizados.

`CostCurves` también incorpora funcionalidades para el trazado de las curvas y el calculo de sus respectivas AUCCs. Además, se le ha añadido opciones para personalizar la apariencia de las gráficas y permitir el trazado de varias curvas a la vez. Esto último es especialmente importante cuando se requiere comparar el rendimiento de varios clasificadores.

La nueva librería `CostCurves` constituye un aporte relevante a la línea de investigación de las Curvas de Coste, ya que permitirá explotar las potencialidades de las curvas y abordar la temática desde un punto de vista práctico. Además podría favorecer a la extensión de su aplicabilidad en otras áreas.

En un futuro, se podría pensar en ampliar la gama de funcionalidades de `CostCurves` y mejorar las ya existentes, en función de nuevos requerimientos y de la retroalimentación que se reciba por parte de los usuarios de la librería.
Para que este paquete se distribuya de forma masiva y este al alcance de un mayor número de usuarios, se podría proponer que la librería pase por el proceso de revisión y posterior publicación en el repositorio oficial de R, el CRAN.

Bibliografía

[1] AHUMADA, H., BAYÁ, A., GRINBLAT, G., AND IZETTA RIERA, C. Extensión de métodos modernos de aprendizaje automatizado y aplicaciones. *XIII Workshop de Investigadores en Ciencias de la Computación* (2011).

[2] ASUNCION, A., AND NEWMAN, D. Machine learning repository. *UCI University of California, Irvine* (2007).

[3] DOMINGOS, P. A few useful things to know about machine learning. *Communications of the ACM 55*, 10 (2012), 78–87.

[4] DRUMMOND, C., AND HOLTE, R. C. Explicitly representing expected cost: An alternative to roc representation. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining* (2000), ACM, pp. 198–207.

[5] DRUMMOND, C., AND HOLTE, R. C. What roc curves can't do (and cost curves can). In *ROCAI* (2004), Citeseer, pp. 19–26.

[6] DRUMMOND, C., AND HOLTE, R. C. Cost curves: An improved method for visualizing classifier performance. *Machine Learning conference, Canadá -Oct* (2006).

[7] FAWCETT, T. Roc graphs: Notes and practical considerations for researchers. *Machine learning 31*, 1 (2004), 1–38.

[8] FAWCETT, T. An introduction to roc analysis. *Pattern recognition letters 27*, 8 (2006), 861–874.

[9] FERRI, C., HERNÁNDEZ-ORALLO, J., AND FLACH, P. A. Brier curves: a new cost-based visualisation of classifier performance. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* (2011), pp. 585–592.

[10] FERRI, C., HERNÁNDEZ-ORALLO, J., AND FLACH, P. A. A coherent interpretation of auc as a measure of aggregated classification performance. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* (2011), pp. 657–664.

[11] FLACH, P. A. The geometry of roc space: understanding machine learning metrics through roc isometrics. In *ICML* (2003), pp. 194–201.

[12] FLACH, P. A. Classification in context: Adapting to changes in class and cost distribution. *LMCE-2014* (2014).

[13] HAY, A. The derivation of global estimates from a confusion matrix. *International Journal of Remote Sensing 9*, 8 (1988), 1395–1398.

[14] HERNÁNDEZ-ORALLO, J., FLACH, P., AND FERRI, C. A unified view of performance metrics: Translating threshold choice into expected classification loss. *Journal of Machine Learning Research 13*, Oct (2012), 2813–2869.

[15] HERNÁNDEZ-ORALLO, J., FLACH, P., AND FERRI, C. Roc curves in cost space. *Machine Learning 93*, 1 (2013), 71–91.

[16] HORNIK, K., KARATZOGLOU, D. M., ZEILEIS, A., AND HORNIK, M. K. The rweka package. *R-CRAN* (2007).

[17] HÜLLERMEIER, E. Fuzzy sets in machine learning and data mining. *Applied Soft Computing 11*, 2 (2011), 1493–1505.

[18] JIMÉNEZ, M. G., AND SIERRA, A. Á. Análisis de datos en weka–pruebas de selectividad. *línea] disponible en http://www. it. uc3m. es/jvillena/irc/practicas/06-07/28. pdf* (2010).

[19] KENDALL, M. G. A new measure of rank correlation. *Biometrika 30*, 1/2 (1938), 81–93.

[20] KOHAVI, R., ET AL. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai* (1995), vol. 14, pp. 1137–1145.

[21] KOTSIANTIS, S. B., ZAHARAKIS, I., AND PINTELAS, P. Supervised machine learning: A review of classification techniques, 2007.

[22] LECUN, Y., BENGIO, Y., AND HINTON, G. Deep learning. *Nature 521*, 7553 (2015), 436–444.

[23] LEISCH, F. Creating r packages: A tutorial. *publisher: Physica Verlag* (2008).

[24] MARSLAND, S. *Machine learning: an algorithmic perspective*. CRC press, 2015.

[25] MARTÍNEZ LADRÓN DE GUEVARA, J. Bioseq: una librería para bioinformática en r. *Universidad Complutense de Madrid* (2013).

[26] METZ, C. E. Basic principles of roc analysis. In *Seminars in nuclear medicine* (1978), vol. 8, Elsevier, pp. 283–298.

[27] MOHRI, M., ROSTAMIZADEH, A., AND TALWALKAR, A. *Foundations of machine learning*. MIT press, 2012.

[28] MONARD, M. C., AND BATISTA, G. E. Learmng with skewed class distrihutions. *Advances in Logic, Artificial Intelligence, and Robotics: LAPTEC 2002 85* (2002), 173.

[29] MORENO, A. *Aprendizaje automático*. Edicions UPC, ISBN: 9788483019962, 1994.

[30] MUENCHEN, B. r4stats. com, 2012.

[31] OLIDEN, P. E. ¿ existe vida más allá del spss? descubre r. *Psicothema 21*, 4 (2009), 652–655.

[32] PROVOST, F. Machine learning from imbalanced data sets 101. In *Proceedings of the AAAI'2000 workshop on imbalanced data sets* (2000), pp. 1–3.

[33] ROBIN, X., TURCK, N., HAINARD, A., TIBERTI, N., LISACEK, F., SANCHEZ, J.-C., AND MÜLLER, M. proc: an open-source package for r and s+ to analyze and compare roc curves. *BMC bioinformatics 12*, 1 (2011), 1.

[34] ROSSI, P. 1 making r packages under windows: A tutorial, 2006.

[35] SALAS, C. ¿ por qué comprar un programa estadístico si existe r. *Ecología austral 18*, 2 (2008), 223–231.

[36] SHALEV-SHWARTZ, S., AND BEN-DAVID, S. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

[37] SING, T., SANDER, O., BEERENWINKEL, N., AND LENGAUER, T. Rocr: visualizing classifier performance in r. *Bioinformatics 21*, 20 (2005), 3940–3941.

[38] SMITH, T. C., AND FRANK, E. *Statistical Genomics: Methods and Protocols*. Springer, New York, NY, 2016, ch. Introducing Machine Learning Concepts with WEKA, pp. 353–378.

[39] SUTTON, R. S., AND BARTO, A. G. *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[40] TEAM, R. C. Writing r extensions. *R Foundation for Statistical Computing* (1999).

[41] UNIVERSITY OF FLORIDA, G. Tutorial on making simple r packages, 2008.

[42] WEBB, G. I., AND TING, K. M. On the application of roc analysis to predict classification performance under varying class distributions. *Machine Learning 58*, 1 (2005), 25–32.

[43] WILSON, G., ARULIAH, D., BROWN, C. T., HONG, N. P. C., DAVIS, M., GUY, R. T., HADDOCK, S. H., HUFF, K. D., MITCHELL, I. M., PLUMBLEY, M. D., ET AL. Best practices for scientific computing. *PLoS Biol 12*, 1 (2014), e1001745.

[44] WITTEN, I. H., AND FRANK, E. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[45] ZHANG, X., JIANG, C., AND LUO, M.-J. Training classifiers for unbalanced distribution and cost-sensitive domains with roc analysis. In *Pacific Rim Knowledge Acquisition Workshop* (2006), Springer, pp. 89–98.

APÉNDICES

APÉNDICE A
Manual de Usuario

Package 'CostCurves'

July 5, 2016

Type Package

Title Display and analyze Cost Curves and treshold choice

Version 0.1.0

Date 2016-04-25

Author Paulina Morillo

Maintainer Paulina Morillo <paumoal@inf.upv.es>

Description Tools for visualization and comparison of cost curves and the choice of threshold. Given predictions and class labels . ROC curve can be plotted and TPR (True positives rate), FPR (FP False positives rate) and AUCC (area under the curve costs) can also be computed.

License GPL-3

LazyData TRUE

Imports flux

RoxygenNote 5.0.1

NeedsCompilation no

R topics documented:

BrierCurves *Plotting Brier Curves*

Description

Function to plot loss against operating condition using the probabilistic choice method (Brier Curves)

Usage

```
BrierCurves(predictions,classes,uniquec=FALSE, loss2skew=FALSE, hold=FALSE,
            plotOFF=FALSE, gridOFF=TRUE, pointsOFF=TRUE, legendOFF=FALSE,
            main, xlab, ylab, namesClassifiers, lwd, lty, col, pch, cex,
            xPosLegend,yPosLegend, cexL)
```

Arguments

predictions	A list of predictions arrays, each array contains scores or predictions of a specific classifier.
classes	A list of classes arrays, each array contains binary classes.
uniquec	If it is TRUE, the same array classes is used for each array in a list predictions.
loss2skew	If it is TRUE, loss by Skew is plotted otherwise loss by cost.
hold	If it is TRUE, the view is maintained to plot a new curve above the current curve.
plotOFF	Disable/enable plot visualization, only return AUC values.
gridOFF	Disable/enable grid visualization.
pointsOFF	Disable/enable point marks visualization.
legendOFF	Disable/enable legend visualization.
main	title.
xlab	x label.
ylab	y label.
namesClassifiers	An array with names of each classifier.
lwd	Line width.
lty	Line type.
col	Line color.
pch	Point type.
cex	Size point.
xPosLegend	x coordinate to be used to position the legend.
yPosLegend	y coordinate to be used to position the legend.
cexL	size of box legend.

Details

Definition:

Function that plots the expected cost/skew against loss. For a given probabilistic classifier and operating condition defined by cost proportion, the probabilistic treshold choice method sets the treshold as fallows:

$$T(c) = c$$

If the operating condition is defined by skew, the treshold is set as fallows:

$$T(z) = z - T(c)z/c$$

The Brier curve for a given classifier is defined as a plot of loss against operating condition using the probabilistic treshold choice method.

In particular, if the operating conditin is determined by cost proportion the Brier curve is defined by:

$$BC(c) = 2(c * pi0(1 - F0(t)) + (1 - c)pi1 * F1(t))$$

A Brier curve for skew is defined by

$$BC(z) = z(1 - F0(t)) + (1 - z)F1(t))$$

Where:

c:	cost values of x_axis between [0, 1].
z:	skew values of x_axis between [0, 1].
t:	treshold t=T(c) or t=T(z) as appropriate.
pi0:	negative class proportion (Y==0)/length(Y).
pi1:	positive class proportion (Y==1)/length(Y).
F1(t):	false positive rate of specific treshold.
1-F0(t):	true positive rate of specific treshold.

Value

An array with AUBC (Area Under Brier Curve) for each test.

Author(s)

Paulina Morillo: <paumoal@inf.upv.es>

References

Ferri, C., Hernandez-orallo, J., & Flach, P. A. (2011). Brier curves: a new cost-based visualisation of classifier performance. In Proceedings of the 28th International Conference on Machine Learning (ICML-11) (pp. 585-592).

See Also

CostCurves, CostLines, KendallCurves, predictions, RateDrivenCurves, RocCurves, TestOptimal, TP_FP.rates, TrainOptimal

Examples

```
#Load data
data(predictions)

#Loss by cost
R<-BrierCurves(list(predictions$A, predictions$B), list(predictions$classes),
uniquec = TRUE, loss2skew = FALSE)


#Loss by skew
R<-BrierCurves(list(predictions$A, predictions$B), list((1-predictions$classes),
predictions$classes), loss2skew = TRUE, gridOFF = FALSE, main=NULL)
```

`CostCurves` *CostCurves*

Description

Function to plot Cost Curves with different option of treshold choice method

Usage

```
CostCurves(predictions, classes, cost_lines = TRUE,
                 test_optimal = TRUE, train_optimal = FALSE,
                 predictionsT = NULL, classesT = NULL,
                 uniquec=FALSE, uniqueTrain=FALSE, uniquecT=FALSE,
                 score_driven = FALSE, rate_driven = FALSE,
                 kendall_curves = FALSE, loss2skew = FALSE,
                 hold = FALSE, gridOFF = TRUE, pointsOFF = TRUE,
                 legendOFF = FALSE,
                 main, xlab, ylab, namesClassifiers, col, lwd,
                 lty, pch, cex,xPosLegend,yPosLegend, cexL)
```

Arguments

`predictions`	A list of predictions or scores arrays.
`classes`	A list of classes arrays, each array contains binary classes.
`cost_lines`	Lines cost are displayed if it is TRUE.
`test_optimal`	Option to plot cost curves using test optimal treshold choice method.
`train_optimal`	Option to plot cost curves using train optimal treshold choice method.
`predictionsT`	A list of classes arrays, each array contains predictions or scores of training classifier. It's necessary only if option train optimal is TRUE.
`classesT`	A list of classes arrays, each array contains binary train classes. It's necessary only if option train_optimal is TRUE.
`uniquec`	If it is TRUE, the same array classes is used for each array in a list predictions.

uniquecT	If it is TRUE, the same array classes is used for each array in a list train predictions. It's ecessary only if the option train_optimal is TRUE.
uniqueTrain	If it is TRUE, the same array of predictionsT and classesT is used for each array in a list predictions. It's necessary only if the option train_optimal is TRUE.
score_driven	If it is TRUE, plot cost curves using score driven treshold choice method.
rate_driven	If it is TRUE, plot cost curves using rate driven treshold choice method.
kendall_curves	If it is TRUE, plot cost curves using kendall driven treshold choice method.
loss2skew	If it is TRUE, loss by Skew is plotted otherwise loss by cost is plotted.
hold	If it is TRUE, the view is maintained to plot a new curve above the current curve.
gridOFF	Disable/enable grid visualization.
pointsOFF	Disable/enable point marks visualization.
legendOFF	Disable/enable legend visualization.
main	title.
xlab	x label.
ylab	y label.
namesClassifiers	An array with names of each classifier.
lwd	Line width.
lty	Line type.
col	Line color.
pch	Point type.
cex	Size point.
xPosLegend	x coordinate to be used to position the legend.
yPosLegend	y coordinate to be used to position the legend.
cexL	Size point. It is posible select any valid option to the graphics parameters of R.

Details

Ploting Cost curves with different treshold choice method.

Cost Lines: CostLines

Test Optimal treshold choice method: TestOptimal

Train Optimal treshold choice method: TrainOptimal

Score Driven treshold choice method: BrierCurves

Rate Driven treshold choice method: RateDrivenCurves

Kendall Curves: KendallCurves

Value

A list of arrays with AUCCs of different cost curves selected.

Author(s)

Paulina Morillo: <paumoal@inf.upv.es>

References

Drummond, C., & Holte, R. C. (2006). Cost curves: An improved method for visualizing classifier performance.

Fawcett, T. (2006). An introduction to ROC analysis. Pattern recognition letters, 27(8), 861-874.

Ferri, C., Hernandez-orallo, J., & Flach, P. A. (2011). Brier curves: a new cost-based visualisation of classifier performance. In Proceedings of the 28th International Conference on Machine Learning (ICML-11) (pp. 585-592).

Hernandez-Orallo, J., Flach, P., & Ferri, C. (2013). ROC curves in cost space. Machine learning, 93(1), 71-91.

See Also

BrierCurves, CostLines, KendallCurves, predictions, RateDrivenCurves, RocCurves, TestOptimal, TP_FP.rates, TrainOptimal

Examples

```
#Load data
data(predictions)

#Loss by Skew
R<-CostCurves(list(predictions$A, predictions$B),
list(predictions$classes), uniquec = TRUE, train_optimal = TRUE,
predictionsT =  list(predictions$B, predictions$A),
classesT = list(predictions$clases, predictions$classes),
loss2skew = TRUE, test_optimal = FALSE,
rate_driven = FALSE, col=list(c("cyan", "red"), c("gray", "blue")),
pointsOFF = FALSE, cex=1)

R<-CostCurves(list(predictions$B), list(predictions$classes),
rate_driven = TRUE, kendall_curves = TRUE, col=c("gray", "red","green"))

#Loss by Cost
R<-CostCurves(list(predictions$A, predictions$B), list(predictions$classes),
uniquec = TRUE,train_optimal = TRUE,
predictionsT =  list(predictions$B, predictions$A),
classesT = list(predictions$classes), uniquecT = TRUE)

R<-CostCurves(list(predictions$A, predictions$B), list(predictions$classes),
uniquec = TRUE, train_optimal = TRUE, predictionsT =  list(predictions$B),
classesT = list(predictions$classes), uniqueTrain = TRUE,
kendall_curves = TRUE)
```

`CostLines` *Plotting Cost Curves based on Cost or Skew*

Description

Function to plot cost curves based on *cost* or *skew*. If interested analysing the influence of class proportion and cost proportion at the same time, it's possible choice *"loss by skew"* option, in this case the *"loss by cost"* curve is normalized.

Usage

```
CostLines(predictions,classes,uniquec=FALSE, loss2skew=FALSE, hold=FALSE,
          gridOFF=TRUE, legendOFF=FALSE, main, xlab, ylab, namesClassifiers,
           lwd, lty, col, xPosLegend,yPosLegend, cexL)
```

Arguments

predictions	A list of predictions arrays, each array contains scores or predictions of a specific classifier.
classes	A list of classes arrays, each array contains binary classes.
uniquec	If it is TRUE, the same array classes is used for each array in a list predictions.
loss2skew	If it is TRUE, loss by Skew is plotted otherwise loss by cost is plotted.
hold	If it is TRUE, the view is maintained to plot a new curve above the current curve.
gridOFF	Disable/enable grid visualization.
legendOFF	Disable/enable legend visualization.
main	title.
xlab	x label.
ylab	y label.
namesClassifiers	An array with names of each classifier.
lwd	Line width. It is posible select any valid option to the graphics parameters of R.
lty	Line type. It is posible select any valid option to the graphics parameters of R.
col	Line color. It is posible select any valid option to the graphics parameters of R.
xPosLegend	x coordinate to be used to position the legend.
yPosLegend	y coordinate to be used to position the legend.
cexL	size of box legend.

Details

The loss which is produced at a decision treshold *t* and a cost proportion *c* is given by formula:

Loss by Cost:

$$2(c * pi0(1 - F0(t)) + (1 - c)pi1 * F1(t))$$

The loss which is produced at a decision treshold *t* and a skew *z* is given by formula:

Loss by Skew:

$$z(1 - F0(t)) + (1 - z)F1(t))$$

Where:

c:	cost values of x_axis between [0, 1].
z:	skew values of x_axis between [0, 1].
t:	treshold.
pi0:	negative class proportion (Y==0)/length(Y).
pi1:	positive class proportion (Y==1)/length(Y).
F1(t):	false positive rate of specific treshold.
1-F0(t):	true positive rate of specific treshold.

Value

No return value.

Author(s)

Paulina Morillo: <paumoal@inf.upv.es>

References

Hernandez-Orallo, J., Flach, P., & Ferri, C. (2013). ROC curves in cost space. Machine learning, 93(1), 71-91.

See Also

BrierCurves, CostCurves, KendallCurves, predictions, RateDrivenCurves, RocCurves, TestOptimal, TP_FP.rates, TrainOptimal

Examples

```
data(predictions)
#Loss by cost
CostLines(list(predictions$A, predictions$B), list(predictions$classes),
uniquec=TRUE)

#Loss by skew
CostLines(list(predictions$A, predictions$B), list(predictions$classes),
uniquec=TRUE, loss2skew = TRUE)

#names legend
CostLines(list(predictions$A, predictions$B), list(predictions$classes,
predictions$classes), loss2skew = TRUE, col=c("blue", "red"), lty=c(1, 2),
            namesClassifiers = c("A","B"))

#LegendOFF
CostLines(list(predictions$A, predictions$B), list(predictions$classes),
uniquec=TRUE, loss2skew = TRUE, legendOFF=TRUE, lty=5)
```

KendallCurves *Plotting Kendall Curves*

Description

Function to plot the expected loss of the model, using the rate driven treshold choice method once the loss of a perfect ranker is discounted

Usage

```
KendallCurves(predictions,classes,uniquec=FALSE, loss2skew=FALSE, hold=FALSE,
               plotOFF=FALSE, gridOFF=TRUE, pointsOFF=TRUE, legendOFF=FALSE,
               main, xlab, ylab, namesClassifiers, lwd, lty, col, pch, cex,
               xPosLegend,yPosLegend, cexL)
```

Arguments

predictions	A list of predictions arrays, each array contains scores or predictions of a specific classifier.
classes	A list of classes arrays, each array contains binary classes.
uniquec	If it is TRUE, the same array classes is used for each array in a list predictions.
loss2skew	If it is TRUE, loss by Skew is plotted otherwise loss by cost is plotted.
hold	If it is TRUE, the view is maintained to plot a new curve above the current curve.
plotOFF	Disable/enable plot visualization, only return AUC values.
gridOFF	Disable/enable grid visualization.
pointsOFF	Disable/enable point marks visualization.
legendOFF	Disable/enable legend visualization.
main	title.
xlab	x label.
ylab	y label.
namesClassifiers	An array with names of each classifier.
lwd	Line width.
lty	Line type.
col	Line color.
pch	Point type.
cex	Size point.
xPosLegend	x coordinate to be used to position the legend.
yPosLegend	y coordinate to be used to position the legend.
cexL	size of box legend.

Details

The Kendall curve is define as fallows:

$$Q(c) = 2pi1 * F1(R^{-}1(c)), ifc <= pi0$$

$$Q(c) = 2pi0(1 - F0(R^{-}1(c))), ifc >= pi0$$

and

$$R(t) = pi0 * F0(t) + pi1 * F1(t), byc$$

$$R(t) = (F0(t) + F1(t))/2, byz$$

Where:

c:	cost values of x_axis between [0, 1].
z:	skew values of x_axis between [0, 1].
t:	treshold t=R^-1(c) or t=R^-1(c) as appropriate, and c=R(t)
pi0:	negative class proportion (Y==0)/length(Y).
pi1:	positive class proportion (Y==1)/length(Y).
F1(t):	false positive rate of specific treshold.
1-F0(t):	true positive rate of specific treshold.
R(c):	recall that the predicted positive rate.

It shows for each cost proportion *c*, the expected loss of the model, once the loss of a perfect ranker is discounted.

Value

An array with AUKC (Area Under Kendall Curve) for each test.

Author(s)

Paulina Morillo: <paumoal@inf.upv.es>

References

Hernandez-Orallo, J., Flach, P., & Ferri, C. (2013). ROC curves in cost space. Machine learning, 93(1), 71-91.

See Also

BrierCurves, CostCurves, CostLines,predictions, RateDrivenCurves, RocCurves, TestOptimal, TP_FP.rates, TrainOptimal

Examples

```
#Load data
data(predictions)

#Loss by cost
R<-KendallCurves(list(predictions$A, predictions$B), list(predictions$classes), uniquec=TRUE,
main="Kendall Curves")

#Loss by skew
R<-KendallCurves(list(predictions$A, predictions$B), list(predictions$classes), uniquec=TRUE,
loss2skew = TRUE, pointsOFF=FALSE)
```

---

predictions *class and predictions data*

Description

This data frame contains the predictions given by 4 classifiers (A, B , C , D) to 10 instances . The first column contains the classes.

Usage

```
predictions
```

Format

A data.frame containing 10 observations of 5 variables.

Source

This predictions values had been obtained from examples of: Ferri, C., Hernandez-orallo, J., & Flach, P. A. (2011). Brier curves: a new cost-based visualisation of classifier performance. In Proceedings of the 28th International Conference on Machine Learning (ICML-11) (pp. 585-592).

See Also

BrierCurves, CostCurves, CostLines, KendallCurves, RateDrivenCurves, RocCurves, TestOptimal, TP_FP.rates, TrainOptimal

Examples

```
#load the dataset
data(predictions)
#table of classes
table(predictions$classes)
#order of predictions
order(predictions$A)
```

RateDrivenCurves *Plotting of lower envelope of Cost Curves*

Description

Function to plot Cost Curves using Rate Driven Treshold Choice

Usage

```
RateDrivenCurves(predictions, classes, uniquec=FALSE, loss2skew=FALSE, hold=FALSE,
                 plotOFF=FALSE, gridOFF=TRUE, pointsOFF=TRUE, legendOFF=FALSE,
                 main, xlab, ylab, namesClassifiers, col, lwd, lty, pch, cex,
                 xPosLegend,yPosLegend, cexL)
```

Arguments

predictions	A list of predictions arrays, each array contains scores or predictions of a specific classifier.
classes	A list of classes arrays, each array contains binary classes.
uniquec	If it is TRUE, the same array classes is used for each array in a list predictions.
loss2skew	If it is TRUE, loss by Skew is plotted otherwise loss by cost is plotted.
hold	If it is TRUE, the view is maintained to plot a new curve above the current curve.
plotOFF	Disable/enable plot visualization, only return AUC values.
gridOFF	Disable/enable grid visualization.
pointsOFF	Disable/enable point marks visualization.
legendOFF	Disable/enable legend visualization.
main	title.
xlab	x label.
ylab	y label.

namesClassifiers	An array with names of each classifier
lwd	Line width.
lty	Line type.
col	Line color.
pch	Point type.
cex	Size point.
xPosLegend	x coordinate to be used to position the legend.
yPosLegend	y coordinate to be used to position the legend.
cexL	size of box legend.

Details

The rate-driven treshold choice method is a natural way of choosing the tresholds, especially when we only have a ranking or a poorly calibrated probabilistic caddifier.

The rate driven cost curves by cost is defined as a plot of:

$$2(c * pi0(1 - F0((R^{-}1(c)))) + (1 - c)pi1 * F1((R^{-}1(c))))$$

on the *axis y* against cost *c*

The rate driven cost curves by skew is defined as a plot of:

$$z(1 - F0(R^{-}1(c))) + (1 - z)F1(R^{-}1(c)))$$

on the *axis y* against skew z

and

$$R(t) = pi0 * F0(t) + pi1 * F1(t), byc$$

$$R(t) = (F0(t) + F1(t))/2, byz$$

Where:

c:	cost values of x_axis between [0, 1].
z:	skew values of x_axis between [0, 1].
t:	treshold t=R^-1(c) or t=R^-1(c) as appropriate, and c=R(t)
pi0:	negative class proportion (Y==0)/length(Y).
pi1:	positive class proportion (Y==1)/length(Y).
F1(t):	false positive rate of specific treshold.
1-F0(t):	true positive rate of specific treshold.
R(c):	recall that the predicted positive rate.

Value

An array with AUKC (Area Under Kendall Curve) for each test.

Author(s)

Paulina Morillo: <paumoal@inf.upv.es>

References

Hernandez-Orallo, J., Flach, P., & Ferri, C. (2013). ROC curves in cost space. Machine learning, 93(1), 71-91.

See Also

BrierCurves, CostCurves, CostLines, KendallCurves, predictions, RocCurves, TestOptimal, TP_FP.rates, TrainOptimal

Examples

```
#Load data
data(predictions)

#Loss by cost
R<-RateDrivenCurves(list(predictions$A, predictions$B),list(predictions$classes), uniquec=TRUE)

#Loss by skew
R<-RateDrivenCurves(list(predictions$A, predictions$B), list(predictions$classes), uniquec=TRUE,
loss2skew = TRUE)
```

RocCurves *Plotting a ROC Curves*

Description

Function to plot ROC Curves by trivial definition TP vs FP

Usage

```
RocCurves(predictions, classes, positiveY=FALSE, plotCH=TRUE,uniquec=FALSE, hold=FALSE,
      plotOFF=FALSE, gridOFF=TRUE, pointsOFF=TRUE, legendOFF=FALSE,
          main, xlab, ylab, namesClassifiers, lwd, lty, col, pch, cex,
          xPosLegend,yPosLegend, cexL)
```

Arguments

predictions	A list of predictions arrays, each array contains scores or predictions of a specific classifier.
classes	A list of classes arrays, each array contains binary classes.
positiveY	Set axix y = FO.
plotCH	It is used to turn off Covex Hull Curve.
uniquec	If it is TRUE, the same array classes is used for each array in a list predictions.
hold	If it is TRUE, the view is maintained to plot a new curve above the current curve.
plotOFF	Disable/enable plot visualization, only return AUC values.
gridOFF	Disable/enable grid visualization.

pointsOFF	Disable/enable point marks visualization.
legendOFF	Disable/enable legend visualization.
main	title.
xlab	x label.
ylab	y label.
namesClassifiers	An array with names of each classifier.
lwd	Line width.
lty	Line type.
col	Line color.
pch	Point type.
cex	Size point.
xPosLegend	x coordinate to be used to position the legend.
yPosLegend	y coordinate to be used to position the legend.
cexL	size of box legend.

Details

ROC graphs are two-dimensional graphs in which *TPR* (True positive rate) is plotted against *FPR* (False positive rate). An ROC graph depicts relative tradeoffs between benefits *true positives* and cost *false positives*.

Value

A list with two arrays, first one with AUC (Area Under ROC Curve) and the second one with AUCH (Area Under Convex Hull) for each classifier.

Author(s)

Paulina Morillo: <paumoal@inf.upv.es>

References

Fawcett, T. (2006). An introduction to ROC analysis. Pattern recognition letters, 27(8), 861-874.

See Also

BrierCurves, CostCurves, CostLines, KendallCurves, predictions, RateDrivenCurves, TestOptimal, TP_FP.rates, TrainOptimal

Examples

```
data(predictions)
#Example1
R<-RocCurves(list((1-predictions$D), predictions$C, predictions$B), list(predictions$classes,
predictions$classes,(1-predictions$classes)), gridOFF=FALSE)
#Example2
R<-RocCurves(list(predictions$B, predictions$A), list(predictions$classes), gridOFF=FALSE,
positiveY = TRUE, uniquec = TRUE)
```

`TestOptimal` *Plotting lower envelope of Cost Curves*

Description

Function to plot of lower envelope of Cost Curves

Usage

```
TestOptimal(predictions,classes,uniquec=FALSE, loss2skew=FALSE, hold=FALSE,
            plotOFF=FALSE, gridOFF=TRUE, pointsOFF=TRUE, legendOFF=FALSE,
            main, xlab, ylab, namesClassifiers, lwd, lty, col, pch, cex,
            xPosLegend,yPosLegend, cexL)
```

Arguments

`predictions`	A list of predictions arrays, each array contains scores or predictions of a specific classifier.
`classes`	A list of classes arrays, each array contains binary classes.
`uniquec`	If it is TRUE, the same array classes is used for each array in a list predictions.
`loss2skew`	If it is TRUE, loss by Skew is plotted otherwise loss by cost is plotted.
`hold`	If it is TRUE, the view is maintained to plot a new curve above the current curve.
`plotOFF`	Disable/enable plot visualization, only return AUC values.
`gridOFF`	Disable/enable grid visualization.
`pointsOFF`	Disable/enable point marks visualization.
`legendOFF`	Disable/enable legend visualization.
`main`	title.
`xlab`	x label.
`ylab`	y label.
`namesClassifiers`	An array with names of each classifier.
`lwd`	Line width.
`lty`	Line type.
`col`	Line color.
`pch`	Point type.
`cex`	Size point.
`xPosLegend`	x coordinate to be used to position the legend.
`yPosLegend`	y coordinate to be used to position the legend.
`cexL`	size of box legend.

Details

The lower envelope at any given c (cost) or z (skew) value is define as the lowest loss value on any of the given cost curves at that c or z. Each segment of lower envelope or test optimal correspond to points on a ROC covex hull.

Value

A list of lists, for each classifier is returned: extreme points of each segment of lower envelope, thresholds and AUCC (area under curve cost).

Author(s)

Paulina Morillo: <paumoal@inf.upv.es>

References

Drummond, C., & Holte, R. C. (2006). Cost curves: An improved method for visualizing classifier performance.

Hernandez-Orallo, J., Flach, P., & Ferri, C. (2013). ROC curves in cost space. Machine learning, 93(1), 71-91.

See Also

BrierCurves, CostCurves, CostLines, KendallCurves, predictions, RateDrivenCurves, RocCurves, TP_FP.rates, TrainOptimal

Examples

```
#Load data
data(predictions)

#Loss by cost
R<-TestOptimal(list(predictions$A, predictions$B), list(predictions$classes), uniquec=TRUE)

#Loss by skew
R<-TestOptimal(list(predictions$A, predictions$B), list(predictions$classes), uniquec=TRUE,
loss2skew = TRUE)

#names legend
R<-TestOptimal(list(predictions$A, predictions$B), list(predictions$classes, predictions$classes),
plotOFF=TRUE)

#LegendOFF
R<-TestOptimal(list(predictions$A, predictions$B), list(predictions$classes), uniquec=TRUE,
loss2skew = TRUE, pointsOFF=FALSE, legendOFF=TRUE, gridOFF=FALSE)
```

TP_FP.rates *Calculates TP and FP rates*

Description

Calculates TP and FP rates, given and any of preds and classes

Usage

```
TP_FP.rates(predictions, classes)
```

Arguments

predictions	An array where each element contains a score or prediction of each instance
classes	A vector with class for each instance

Details

We calculate a *TPR* (True positive rate) and *FPR* (False positive rate) rates for each treshold between each pair of scores or predictions values.

We consider "1" for positives class (P), and "0" for negatives class (N)

TP:	Number of true positives.
FP:	Number of false positives.
TPR:	Estimate as: TP/P.
FPR:	Estimate as: FP/N.

Value

An array with two columns, first one corresponding to TPR and second one corresponding to FPR.

Author(s)

Paulina Morillo: <paumoal@inf.upv.es>

References

Drummond, C., & Holte, R. C. (2006). Cost curves: An improved method for visualizing classifier performance.

See Also

BrierCurves, CostCurves, CostLines, KendallCurves, predictions, RateDrivenCurves, RocCurves, TestOptimal, TrainOptimal

Examples

```
predictions <- round(runif(10), 0)
classes <- round(runif(10), 1)
TP_FP.rates(predictions, classes)
```

TrainOptimal *Plotting Cost Curves using Optimal Training treshold choice*

Description

Function to plot optimal training curve

Usage

```
TrainOptimal(predictions_train, classes_train, predictions_test, classes_test,
                uniquecT=FALSE, uniquect=FALSE, refuseT=FALSE, loss2skew=FALSE,
             hold=FALSE, plotOFF=FALSE, pointsOFF=TRUE, gridOFF=TRUE,legendOFF=FALSE,
               main, xlab, ylab, namesClassifiers,namesTests, lwd, lty, col,pch,
                cex, xPosLegend,yPosLegend, cexL)
```

Arguments

predictions_train	A list of predictions arrays, each array contains scores or probabilities of a specific classifier training.
classes_train	A list of classes arrays, each array contains binary classes of training.
predictions_test	A list of predictions arrays, each array contains scores test.
classes_test	A list of classes arrays, each array contains binary classes test.
uniquecT	If it is TRUE, the same array classes is used for each array in a list predictions Training.
uniquect	If it is TRUE, the same array classes is used for each array in a list predictions test.
refuseT	It is possible to use the same training classifier for every test.
loss2skew	If it is TRUE, loss by Skew is plotted otherwise loss by cost is plotted.
hold	If it is TRUE, the view is maintained to plot a new curve above the current curve.
plotOFF	Disable/enable plot visualization, only return AUC values.
gridOFF	Disable/enable grid visualization.
pointsOFF	Disable/enable point marks visualization.
legendOFF	Disable/enable legend visualization.
main	title.
xlab	x label.
ylab	y label.
namesClassifiers	An array with names of each classifier.
namesTests	An array with names of each classifier.
lwd	Line width.
lty	Line type.
col	Line color.
pch	Point type.
cex	Size point.
xPosLegend	x coordinate to be used to position the legend.
yPosLegend	y coordinate to be used to position the legend.
cexL	size of box legend.

Details

The similar idea over *testOptimal*. This function plot the cost curves using a optimal treshold train on a test.

Value

An array with AUCC (Area Under Cost Curve) for each test.

Author(s)

Paulina Morillo: <paumoal@inf.upv.es>

References

#It reference is not exact. Hernandez-Orallo, J., Flach, P., & Ferri, C. (2013). ROC curves in cost space. Machine learning, 93(1), 71-91.

See Also

BrierCurves, CostCurves, CostLines, KendallCurves, predictions, RateDrivenCurves, RocCurves, TestOptimal, TP_FP.rates

Examples

```
#Load data
data(predictions)

#Loss by Skew
R<-TrainOptimal(list(predictions$A), list(predictions$classes),
list(predictions$A, predictions$B), list(predictions$classes),
uniquect = TRUE, uniquecT = TRUE, loss2skew = TRUE, refuseT = TRUE)

#Loss by Cost
R<-TrainOptimal(list(predictions$A, predictions$B), list(predictions$classes),
list(predictions$B, predictions$A), list(predictions$classes), uniquect = TRUE,
namesClassifiers=c("A", "B"), namesTests=c("B","A"), uniquecT = TRUE)
```

Index

APÉNDICE B
Código en R: Casos de Uso

Casos de Uso: Curvas de Coste

Paulina Morillo

Librerias utilizadas

```
library("RWeka")
```

```
## Warning: package 'RWeka' was built under R version 3.2.5
```

```
library("caret")
```

```
## Warning: package 'caret' was built under R version 3.2.5

## Loading required package: lattice

## Warning: package 'lattice' was built under R version 3.2.5

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.2.5
```

```
library("CostCurves")
```

Caso 1: Desarrollo y evaluación de un clasificador

Tratamiento de datos

```
#Carga de Datos
data <- read.arff("credit-a.arff")
data[seq(10),]
```

```
##    A1    A2     A3 A4 A5 A6 A7    A8 A9 A10 A11 A12 A13 A14   A15 class
## 1   b 30.83  0.000  u  g  w  v 1.250  t   t   1   f   g 202     0     +
## 2   a 58.67  4.460  u  g  q  h 3.040  t   t   6   f   g  43   560     +
## 3   a 24.50  0.500  u  g  q  h 1.500  t   f   0   f   g 280   824     +
## 4   b 27.83  1.540  u  g  w  v 3.750  t   t   5   t   g 100     3     +
## 5   b 20.17  5.625  u  g  w  v 1.710  t   f   0   f   s 120     0     +
## 6   b 32.08  4.000  u  g  m  v 2.500  t   f   0   t   g 360     0     +
## 7   b 33.17  1.040  u  g  r  h 6.500  t   f   0   t   g 164 31285     +
## 8   a 22.92 11.585  u  g cc  v 0.040  t   f   0   f   g  80  1349     +
## 9   b 54.42  0.500  y  p  k  h 3.960  t   f   0   f   g 180   314     +
## 10  b 42.50  4.915  y  p  w  v 3.165  t   f   0   t   g  52  1442     +
```

```r
summary(data)
```

```
##     A1            A2              A3            A4         A5
##  b   :468   Min.   :13.75   Min.   : 0.000   u   :519   g   :519
##  a   :210   1st Qu.:22.60   1st Qu.: 1.000   y   :163   p   :163
##  NA's: 12   Median :28.46   Median : 2.750   l   :  2   gg  :  2
##             Mean   :31.57   Mean   : 4.759   t   :  0   NA's:  6
##             3rd Qu.:38.23   3rd Qu.: 7.207   NA's:  6
##             Max.   :80.25   Max.   :28.000
##             NA's   :12
##        A6           A7            A8         A9      A10
##  c      :137   v      :399   Min.   : 0.000   t:361   t:295
##  q      : 78   h      :138   1st Qu.: 0.165   f:329   f:395
##  w      : 64   bb     : 59   Median : 1.000
##  i      : 59   ff     : 57   Mean   : 2.223
##  aa     : 54   j      :  8   3rd Qu.: 2.625
##  (Other):289   (Other): 20   Max.   :28.500
##  NA's   :  9   NA's   :  9
##       A11       A12     A13          A14            A15           class
##  Min.   : 0.0   t:316   g:625   Min.   :   0   Min.   :     0.0   +:307
##  1st Qu.: 0.0   f:374   p:  8   1st Qu.:  75   1st Qu.:     0.0   -:383
##  Median : 0.0           s: 57   Median : 160   Median :     5.0
##  Mean   : 2.4                   Mean   : 184   Mean   :  1017.4
##  3rd Qu.: 3.0                   3rd Qu.: 276   3rd Qu.:   395.5
##  Max.   :67.0                   Max.   :2000   Max.   :100000.0
##                                 NA's   :13
```

```r
# La última columna corresponde a la clase de cada instancia
posParamEstudio<-length(data[1,])
nomParamEstudio<-names(data)[posParamEstudio]
muestra<-sample(seq(688/2),10)

# Ordenacion de datos
w<-data[sample(nrow(data)),]
tam<-length(data[,1])

# clmin => clase minoritaria
clmin<-unique(data[nomParamEstudio])
if(sum(data[,nomParamEstudio]==clmin[[1]][1])>(tam/2)){
  mnom=clmin[[1]][1]
} else mnom=clmin[[1]][2]

# Partición del dataset
indiceStrat<-createDataPartition(w[,posParamEstudio], p = 0.5,list=FALSE)
train<-w[indiceStrat,] # Conjunto de datos de entrenamiento
test<-w[-indiceStrat,] # Conjunto de datos de prueba
```

Creación y entrenamiento del modelo

Modelo J48

```
# ob = > Variable a predecir (Objetivo de aprendizaje)
ob<-paste(nomParamEstudio,"~.",sep="")

# Selección y Entrenamiento del modelo (Modelo J48)
model<- J48(ob, data = train, control = Weka_control(U = TRUE,A=TRUE))

# Resumen del modelo J48
summary(model)
```

```
##
## === Summary ===
##
## Correctly Classified Instances         313               95.7187 %
## Incorrectly Classified Instances        14                4.2813 %
## Kappa statistic                          0.9138
## Mean absolute error                      0.1598
## Root mean squared error                  0.2224
## Relative absolute error                 32.1686 %
## Root relative squared error             44.6386 %
## Total Number of Instances              327
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  143   7 |   a = +
##    7 170 |   b = -
```

```
# Modelo J48
model
```

```
## J48 unpruned tree
## ------------------
##
## A9 = t
## |   A15 <= 375
## |   |   A10 = t
## |   |   |   A6 = c
## |   |   |   |   A2 <= 22.92: - (3.0/1.0)
## |   |   |   |   A2 > 22.92: + (7.0)
## |   |   |   A6 = d: + (1.0)
## |   |   |   A6 = cc: + (9.0)
## |   |   |   A6 = i: + (3.0/1.0)
## |   |   |   A6 = j: + (1.0)
## |   |   |   A6 = k: - (2.0)
## |   |   |   A6 = m: + (1.0)
## |   |   |   A6 = r: + (0.0)
## |   |   |   A6 = q
## |   |   |   |   A1 = b
```

```
## |   |   |   |   |   A3 <= 5.125: + (2.0)
## |   |   |   |   |   A3 > 5.125: - (2.0)
## |   |   |   |   A1 = a: + (4.0)
## |   |   |   A6 = w
## |   |   |   |   A12 = t
## |   |   |   |   |   A3 <= 4: - (2.0)
## |   |   |   |   |   A3 > 4: + (2.0)
## |   |   |   |   A12 = f: + (3.0)
## |   |   |   A6 = x: + (5.0)
## |   |   |   A6 = e: + (3.0/1.0)
## |   |   |   A6 = aa: + (2.0)
## |   |   |   A6 = ff: - (1.0)
## |   |   A10 = f
## |   |   |   A7 = v
## |   |   |   |   A15 <= 0
## |   |   |   |   |   A14 <= 60: + (4.0)
## |   |   |   |   |   A14 > 60
## |   |   |   |   |   |   A6 = c: - (5.0/1.0)
## |   |   |   |   |   |   A6 = d: - (0.0)
## |   |   |   |   |   |   A6 = cc: - (0.0)
## |   |   |   |   |   |   A6 = i: - (0.0)
## |   |   |   |   |   |   A6 = j: - (0.0)
## |   |   |   |   |   |   A6 = k: + (2.0/1.0)
## |   |   |   |   |   |   A6 = m: - (3.0)
## |   |   |   |   |   |   A6 = r: - (0.0)
## |   |   |   |   |   |   A6 = q: + (2.0/1.0)
## |   |   |   |   |   |   A6 = w: + (1.0)
## |   |   |   |   |   |   A6 = x: - (0.0)
## |   |   |   |   |   |   A6 = e: - (0.0)
## |   |   |   |   |   |   A6 = aa
## |   |   |   |   |   |   |   A3 <= 2.5: - (3.0)
## |   |   |   |   |   |   |   A3 > 2.5
## |   |   |   |   |   |   |   |   A3 <= 5: + (2.0)
## |   |   |   |   |   |   |   |   A3 > 5: - (3.0/1.0)
## |   |   |   |   |   |   A6 = ff: - (0.0)
## |   |   |   |   A15 > 0: - (5.0)
## |   |   |   A7 = h
## |   |   |   |   A6 = c: + (3.0)
## |   |   |   |   A6 = d: + (1.0)
## |   |   |   |   A6 = cc: + (1.0)
## |   |   |   |   A6 = i: + (2.0)
## |   |   |   |   A6 = j: + (0.0)
## |   |   |   |   A6 = k: + (0.0)
## |   |   |   |   A6 = m: + (0.0)
## |   |   |   |   A6 = r: + (0.0)
## |   |   |   |   A6 = q: + (3.0)
## |   |   |   |   A6 = w: + (1.0)
## |   |   |   |   A6 = x: - (1.0)
## |   |   |   |   A6 = e: + (1.0)
## |   |   |   |   A6 = aa: + (0.0)
## |   |   |   |   A6 = ff: + (0.0)
## |   |   |   A7 = bb
## |   |   |   |   A14 <= 160: + (4.0/1.0)
## |   |   |   |   A14 > 160: - (2.0)
```

```
## |   |   |   A7 = j: - (1.0)
## |   |   |   A7 = n: + (0.0)
## |   |   |   A7 = z: + (0.0)
## |   |   |   A7 = dd: + (1.0)
## |   |   |   A7 = ff: - (2.0)
## |   |   |   A7 = o: + (0.0)
## |   A15 > 375: + (70.0)
## A9 = f
## |   A13 = g
## |   |   A10 = t: - (32.0)
## |   |   A10 = f
## |   |   |   A6 = c
## |   |   |   |   A1 = b
## |   |   |   |   |   A4 = u
## |   |   |   |   |   |   A15 <= 0: + (3.0/1.0)
## |   |   |   |   |   |   A15 > 0: - (4.0)
## |   |   |   |   |   A4 = y: - (4.0)
## |   |   |   |   |   A4 = l: - (0.0)
## |   |   |   |   |   A4 = t: - (0.0)
## |   |   |   |   A1 = a: - (4.0)
## |   |   |   A6 = d: - (6.0/1.0)
## |   |   |   A6 = cc: - (4.0/1.0)
## |   |   |   A6 = i: - (17.0)
## |   |   |   A6 = j: - (4.0)
## |   |   |   A6 = k: - (12.0)
## |   |   |   A6 = m: - (2.0)
## |   |   |   A6 = r: - (1.0)
## |   |   |   A6 = q: - (4.0/1.0)
## |   |   |   A6 = w
## |   |   |   |   A1 = b
## |   |   |   |   |   A15 <= 4: - (4.0)
## |   |   |   |   |   A15 > 4: + (3.0/1.0)
## |   |   |   |   A1 = a: - (3.0)
## |   |   |   A6 = x: - (2.0)
## |   |   |   A6 = e: - (2.0)
## |   |   |   A6 = aa: - (11.0)
## |   |   |   A6 = ff: - (11.0)
## |   A13 = p: + (1.0)
## |   A13 = s
## |   |   A4 = u
## |   |   |   A6 = c: - (2.0)
## |   |   |   A6 = d: - (1.0)
## |   |   |   A6 = cc: - (0.0)
## |   |   |   A6 = i: - (2.0)
## |   |   |   A6 = j: - (0.0)
## |   |   |   A6 = k: - (3.0)
## |   |   |   A6 = m: - (2.0)
## |   |   |   A6 = r: - (0.0)
## |   |   |   A6 = q: - (1.0)
## |   |   |   A6 = w: - (0.0)
## |   |   |   A6 = x: + (1.0)
## |   |   |   A6 = e: - (0.0)
## |   |   |   A6 = aa: - (0.0)
## |   |   |   A6 = ff: - (1.0)
```

```
## |   |   A4 = y: - (3.0/1.0)
## |   |   A4 = l: + (1.0)
## |   |   A4 = t: - (0.0)
##
## Number of Leaves  :  100
##
## Size of the tree :   127
```

```
# Cálculo de las probabilidades y clases del conjunto de entrenamiento
allprobstrain<-predict(model, newdata = train,type=c("probability"))
probstrain<-allprobstrain[,mnom]

cltrain<-c()
for (i in 1:length(train[,1]))
{
  if(train[i,nomParamEstudio]==mnom)  {cltrain[i]<-1 }
  else cltrain[i]<-0
}

# Resultados del entrenamiento
# (predicciones: cltrain => clases, probtrain => probabilidades)
inptrain<-cbind(cltrain,probstrain)
predicciones_entrenamiento <-inptrain[order(inptrain[,2],inptrain[,1]),]

probs_train<-t(inptrain[,2])
class_train<-t(inptrain[,1])

# Muestra de los resultados del entrenamiento del modelo
print(predicciones_entrenamiento[muestra,])
```

```
##     cltrain probstrain
## 229       0 0.25000000
## 28        0 0.33333333
## 647       1 0.83333333
## 611       1 0.92857143
## 256       1 0.83333333
## 5         0 0.33333333
## 564       0 0.20000000
## 76        1 0.75000000
## 156       0 0.33333333
## 161       0 0.01388889
```

Evaluacion del modelo

```
# Computo de las probabilidades y las clases para el conjunto de datos de prueba
allprobs<-predict(model, newdata = test,type=c("probability"))
probstest<-allprobs[,mnom]

cltest<-c()
for (i in 1:length(test[,1]))
{
```

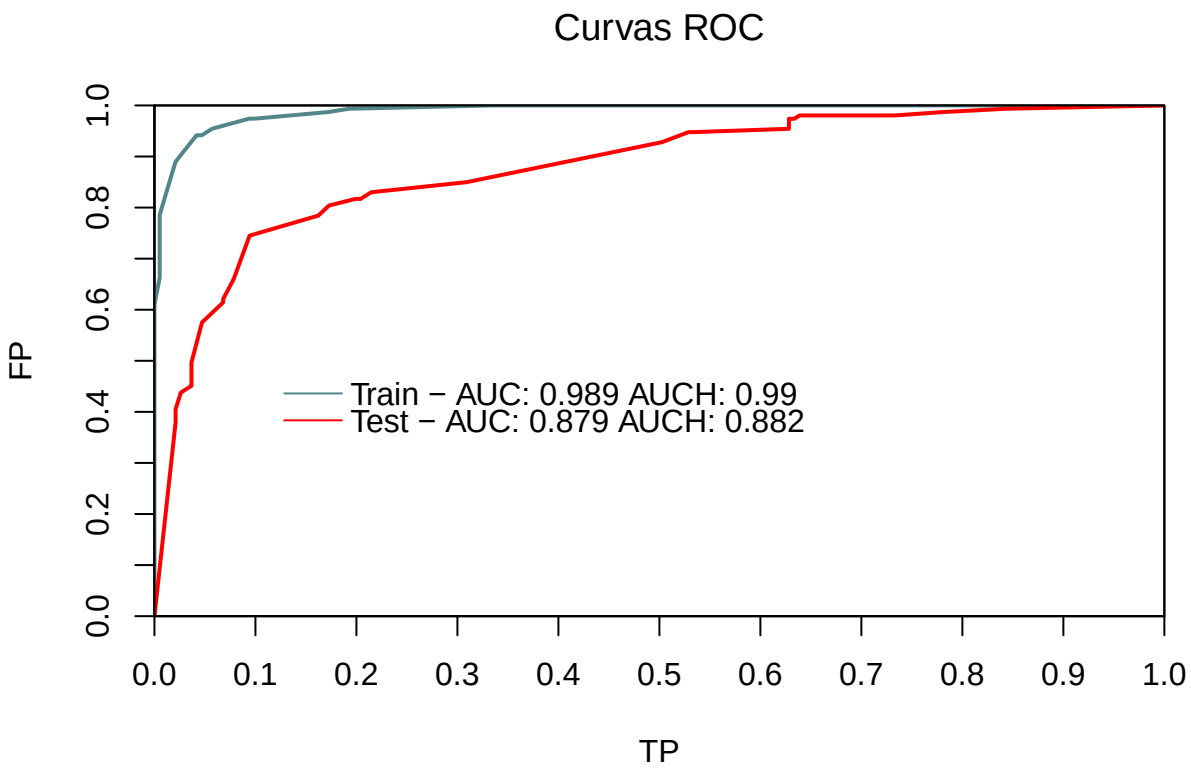
```
  if(test[i,posParamEstudio]==mnom)
  {cltest[i]<-1 }
  else cltest[i]<-0
}

# Resultados de la prueba
# (predicciones: cltest => clases, probtest => probabilidades)
inp<-cbind(cltest,probstest)
predicciones_prueba<-inp[order(inp[,2],inp[,1]),]

probs_test<-t(inp[,2])
class_test<-t(inp[,1])

# Muestra de los resultados de la prueba
print(predicciones_prueba[muestra,])
```

```
##     cltest  probstest
## 89       1 0.25000000
## 131      0 0.40000000
## 74       1 0.80000000
## 640      1 0.92857143
## 684      1 0.75000000
## 243      0 0.33333333
## 566      0 0.25000000
## 476      1 0.75000000
## 219      0 0.40000000
## 7        0 0.01388889
```

Curvas de Coste: Visulización y análisis del rendimiento de un clasificador

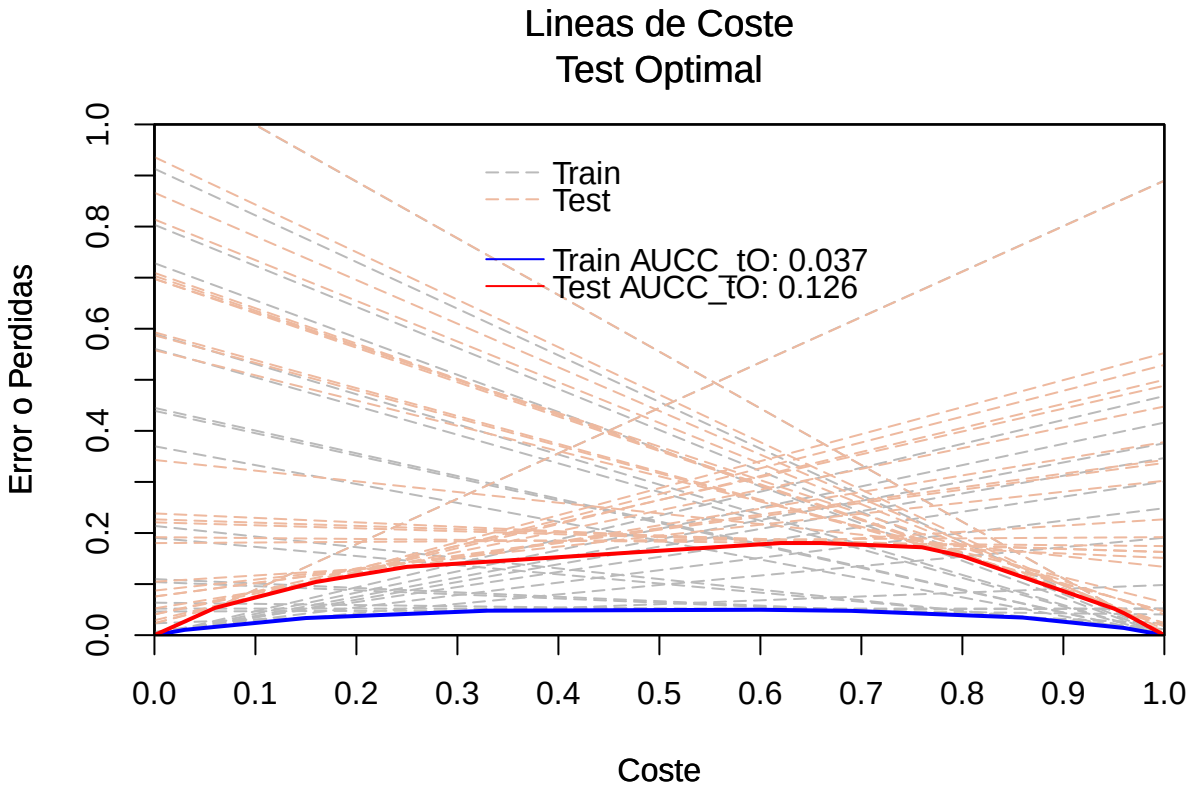
Curvas ROC

```
# Curvas ROC
R <- RocCurves(list(probs_train, probs_test), list(class_train, class_test),
               plotCH = FALSE, main = "Curvas ROC", xlab = "TP", ylab = "FP",
               namesClassifiers = c("Train - ","Test - "),
               xPosLegend = 0.1, yPosLegend = 0.5, cexL = 1)
```



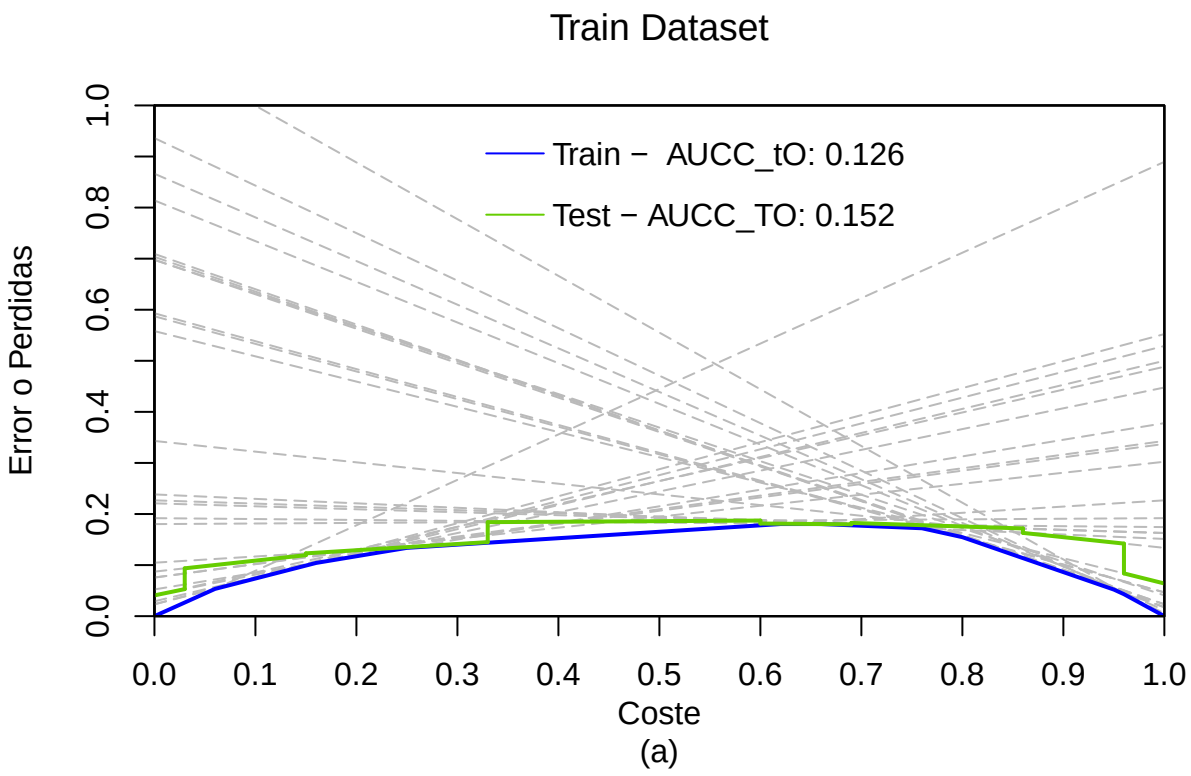
Lineas de Coste y Test Optimal

```
# Lineas de Coste y Test Optimal

#Se pueden superponer graficas, utilizando el prametro hold
R <- CostLines(list(probs_train, probs_test), list(class_train, class_test),
               main = "Lineas de Coste\nTest Optimal", xlab = "Coste",
               ylab = "Error o Perdidas", lwd=1,
               namesClassifiers = c("Train","Test"), cexL=1, xPosLegend = 0.3)

R <- TestOptimal(list(probs_train, probs_test), list(class_train, class_test),
               main = NULL, xlab = NULL,
               ylab = NULL, namesClassifiers = c("Train","Test"), cexL=1,
               xPosLegend = 0.3, yPosLegend = 0.8, hold = TRUE)
```



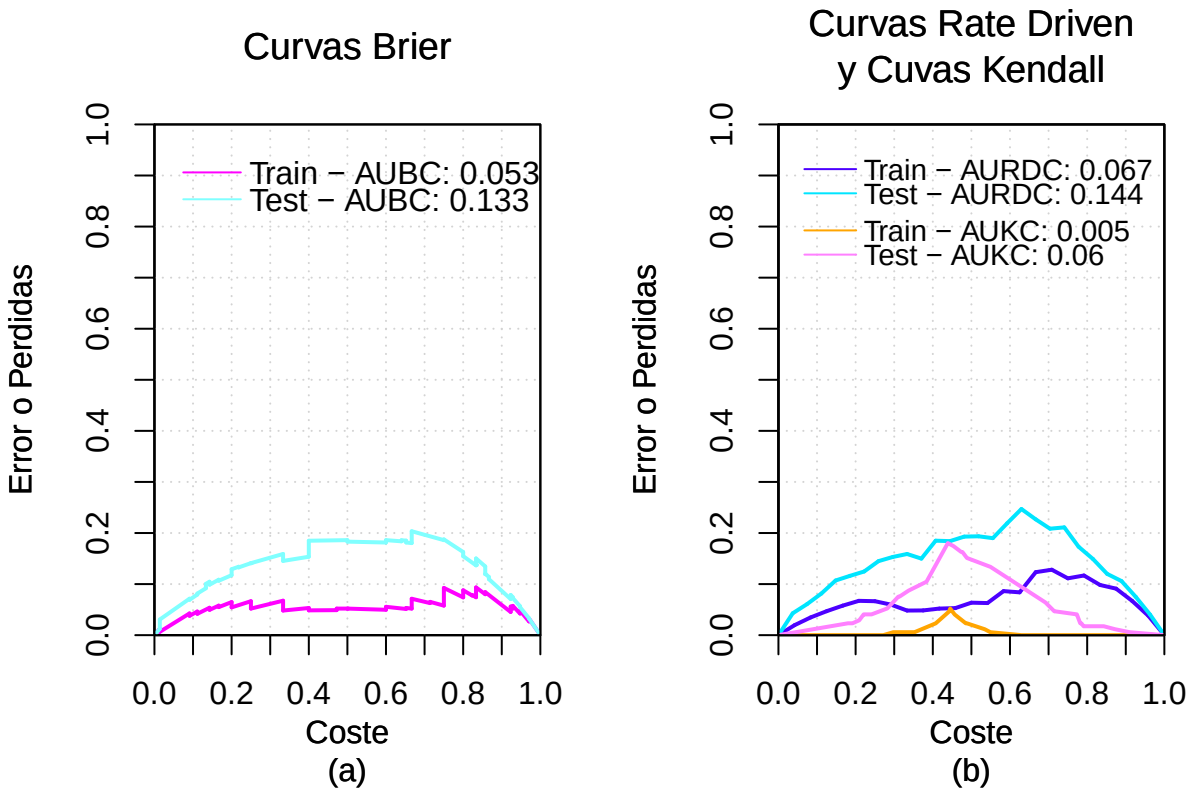
Train Optimal

```
# Train Optimal

R <- CostLines(list(probs_test), list(class_test),
               main = "Train Dataset", xlab = "Coste\n(a)",
               ylab = "Error o Perdidas", legendOFF = TRUE, lwd=1)

R <- TestOptimal(list(probs_test), list(class_test), main = NULL, xlab = NULL,
                 ylab = NULL, hold = TRUE, namesClassifiers = "Train - ",
                 cexL = 1, xPosLegend = 0.3)

R <- TrainOptimal(list(probs_train), list(class_train), list(probs_test),
                  list(class_test), namesClassifiers = "Test -", cexL = 1,
                  xPosLegend = 0.3, yPosLegend = 0.85, hold = TRUE, main = NULL,
                  xlab = NULL, ylab = NULL)
```



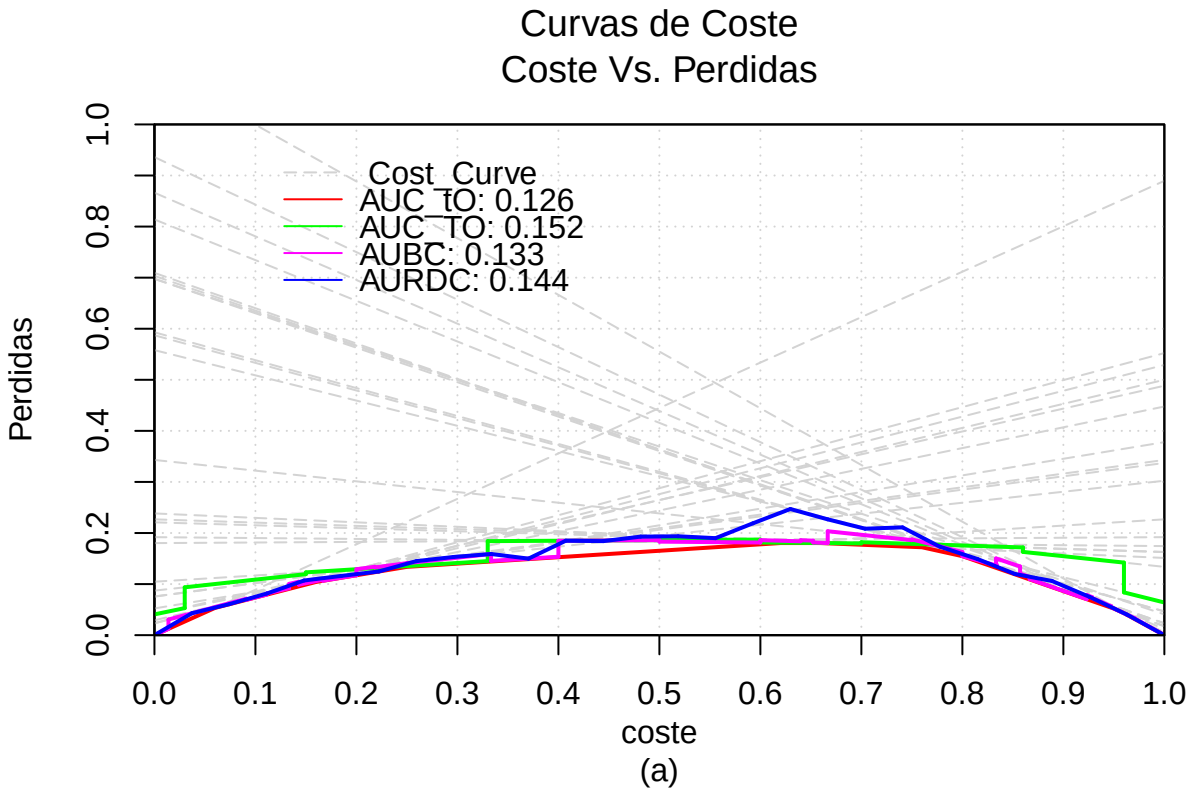
Score driven (Curvas Brier), Rate Driven y Curvas Kendall

```
par(mfrow=c(1,2))

R <- BrierCurves(list(probs_train, probs_test), list(class_train, class_test),
                 main = "Curvas Brier", xlab = "Coste\n(a)", cexL=1,
                 xPosLegend = 0,ylab = "Error o Perdidas",
                 namesClassifiers =  c("Train -", "Test -"),
                 gridOFF = FALSE)

R <- RateDrivenCurves(list(probs_train, probs_test),
                      list(class_train, class_test),
                      main = "Curvas Rate Driven\ny Cuvas Kendall",
                      xlab = "Coste\n(b)", ylab = "Error o Perdidas",
                      namesClassifiers = c("Train -", "Test -"),
                      cexL=0.9, xPosLegend = 0, gridOFF = FALSE)

R <- KendallCurves(list(probs_train, probs_test), list(class_train, class_test),
                   main = NULL, xlab = NULL, ylab = NULL, cexL=0.9,
                   xPosLegend = 0, yPosLegend = 0.85 ,
                   gridOFF = FALSE, hold=TRUE,
                   namesClassifiers = c("Train -", "Test -"))
```

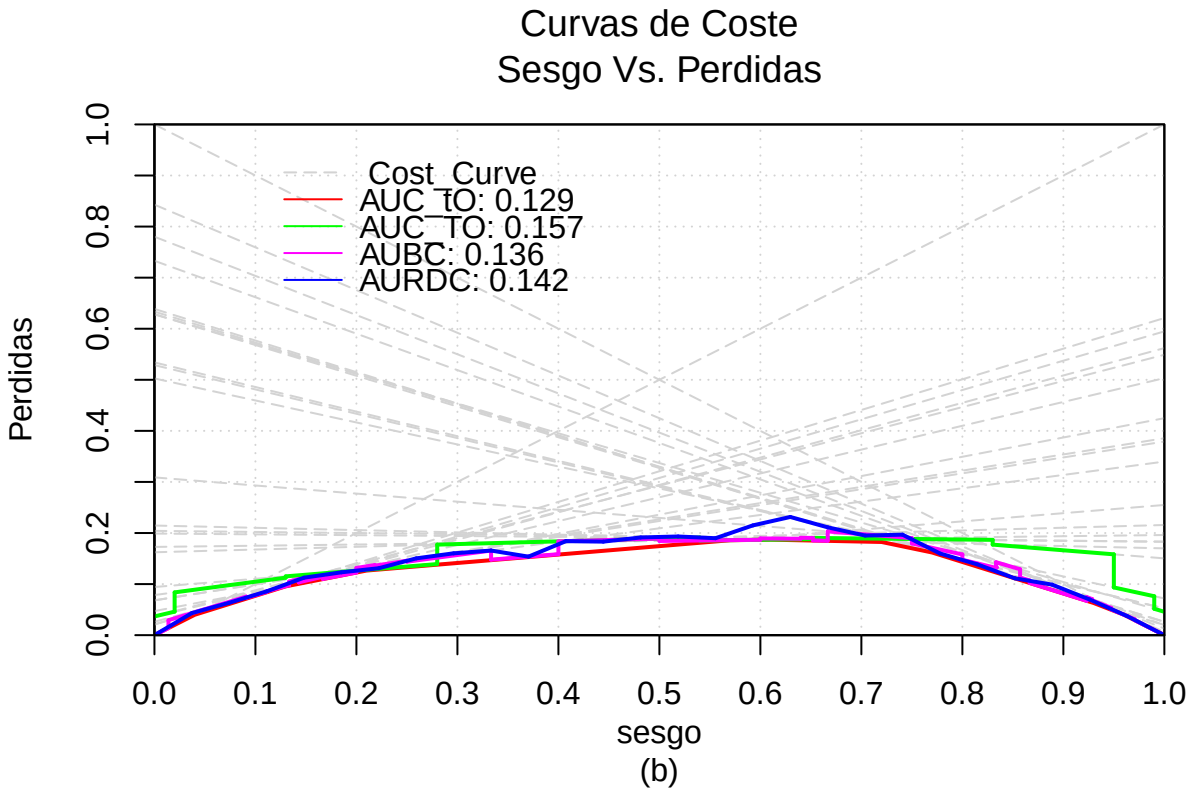


Curvas de Coste en función del coste y en función del sesgo

```
#En funcion del coste
R <- CostCurves(list(probs_test), list(class_test), predictionsT =  list(probs_train),
                classesT =  list(class_train), train_optimal = TRUE,
                score_driven = TRUE, rate_driven = TRUE, namesClassifiers = NULL,
                cexL = 1, xPosLegend = 0.1,
                main ="Curvas de Coste\nCoste Vs. Perdidas", xlab = "coste\n(a)",
                ylab = "Perdidas", lty = c(5, rep(1, 4)),lwd=c(1,rep(2,4)),
                col=c("lightgray", "red", "green", "magenta","blue"))
```




```r
#En funcion del sesgo
R <- CostCurves(list(probs_test), list(class_test),
                predictionsT =  list(probs_train),
                classesT =  list(class_train), train_optimal = TRUE,
                score_driven = TRUE, rate_driven = TRUE, namesClassifiers = NULL,
                cexL = 1, xPosLegend = 0.1,
                main ="Curvas de Coste\nSesgo Vs. Perdidas", xlab = "sesgo\n(b)",
                ylab = "Perdidas", lty = c(5, rep(1, 4)), lwd=c(1,rep(2,4)),
                col=c("lightgray", "red", "green", "magenta","blue"),
                loss2skew = TRUE)
```



Caso2: Comparación del rendimiento de 2 clasificadores

Creación y entrenamiento de un nuevo modelo clasificación

Modelo de Regresión Logística

```
# Selección y Entrenamiento del modelo (Regresion Logistica)
model <- Logistic(ob, data = train)

# Modelo de Regresion Logistica
summary(model)
```

```
##
## === Summary ===
##
## Correctly Classified Instances         290               88.685  %
## Incorrectly Classified Instances        37               11.315  %
## Kappa statistic                          0.7732
## Mean absolute error                      0.1669
## Root mean squared error                  0.2867
## Relative absolute error                 33.613  %
## Root relative squared error             57.5274 %
```

```
## Total Number of Instances              327
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  136  14 |   a = +
##   23 154 |   b = -
```

```
# Cálculo de las probabilidades y clases del conjunto de entrenamiento
allprobstrain<-predict(model, newdata = train,type=c("probability"))
probstrain<-allprobstrain[,mnom]

cltrain<-c()
for (i in 1:length(train[,1]))
{
  if(train[i,nomParamEstudio]==mnom)  {cltrain[i]<-1 }
  else cltrain[i]<-0
}

# Resultados del entrenamiento
# (predicciones: cltrain => clases, probtrain => probabilidades)
inptrain<-cbind(cltrain,probstrain)
predicciones_entrenamiento <-inptrain[order(inptrain[,2],inptrain[,1]),]

probs_train2<-t(inptrain[,2])
class_train2<-t(inptrain[,1])

# Muestra de los resultados del entrenamiento del modelo
print(predicciones_entrenamiento[muestra,])
```

```
##     cltrain probstrain
## 115       1 0.19398058
## 58        0 0.28982809
## 647       1 0.96321677
## 385       1 0.98802322
## 462       1 0.95729999
## 513       0 0.25941774
## 180       0 0.17256229
## 613       1 0.90196115
## 220       0 0.29277135
## 238       0 0.00420003
```

Evaluacion del modelo

```
# Computo de las probabilidades y las clases para el conjunto de datos de prueba
allprobs<-predict(model, newdata = test,type=c("probability"))
probstest<-allprobs[,mnom]

cltest<-c()
for (i in 1:length(test[,1]))
{
```

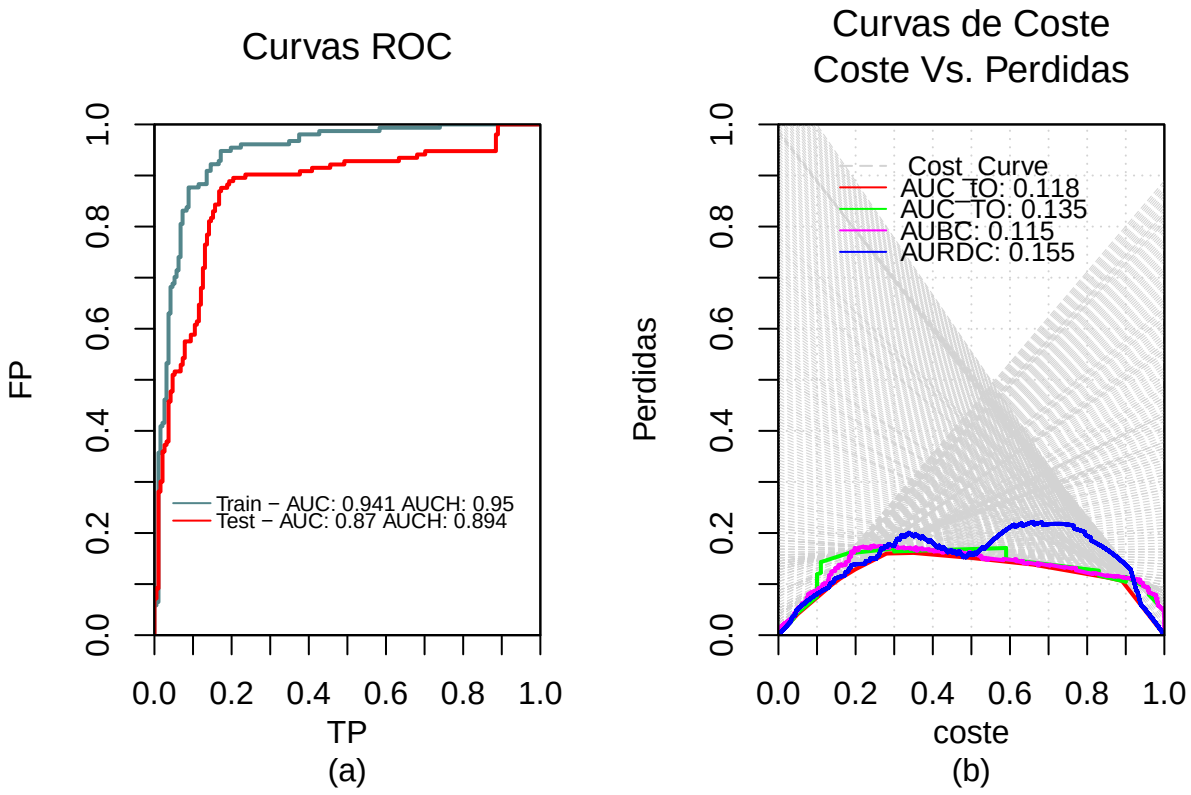
```
  if(test[i,posParamEstudio]==mnom)
  {cltest[i]<-1 }
  else cltest[i]<-0
}

# Resultados de la prueba
# (predicciones: cltest => clases, probtest => probabilidades)
inp<-cbind(cltest,probstest)
predicciones_prueba<-inp[order(inp[,2],inp[,1]),]

probs_test2<-t(inp[,2])
class_test2<-t(inp[,1])

# Muestra de los resultados de la prueba
print(predicciones_prueba[muestra,])
```

```
##     cltest   probstest
## 532      1 0.197718522
## 507      0 0.280130465
## 481      1 0.969321573
## 355      1 0.988736224
## 403      1 0.961226674
## 176      0 0.246972993
## 540      1 0.188769293
## 614      1 0.938244646
## 596      0 0.302326848
## 140      0 0.004711787
```

Visualización de rendimiento de modelo de Regresión Logística

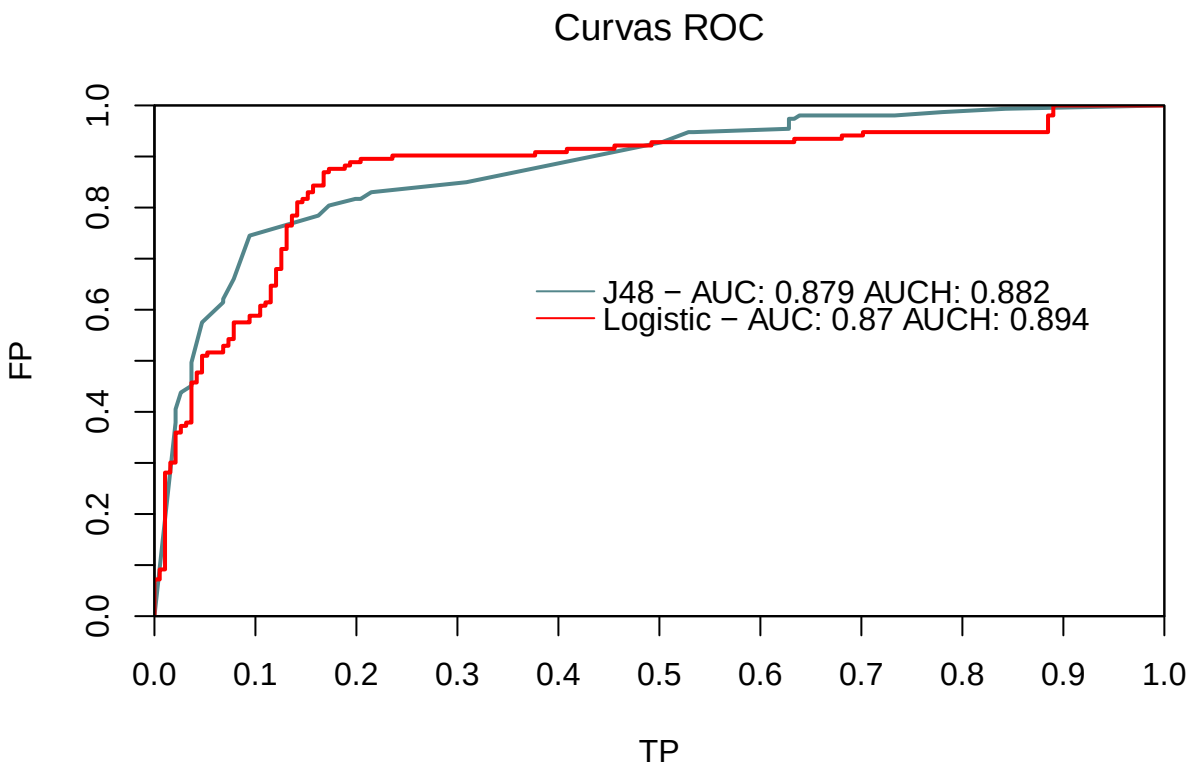
Curvas ROC vs. Curvas de Coste

```
# Curvas ROC
par(mfrow=c(1,2))
R <- RocCurves(list(probs_train2, probs_test2), list(class_train2, class_test2),
               plotCH = FALSE, main = "Curvas ROC", xlab = "TP\n(a)",
               ylab = "FP", namesClassifiers = c("Train - ","Test - "),
               xPosLegend = 0, yPosLegend = 0.3, cexL = 0.65)

R <- CostCurves(list(probs_test2), list(class_test2),
                predictionsT =  list(probs_train2),
                classesT =  list(class_train2), train_optimal = TRUE,
                score_driven = TRUE, rate_driven = TRUE, namesClassifiers = NULL,
                cexL = 0.8, xPosLegend = 0.1,
                main ="Curvas de Coste\nCoste Vs. Perdidas", xlab = "coste\n(b)",
                ylab = "Perdidas", lty = c(5, rep(1, 4)), lwd=c(0.5, rep(2, 4)),
                col=c("lightgray", "red", "green", "magenta","blue"))
```

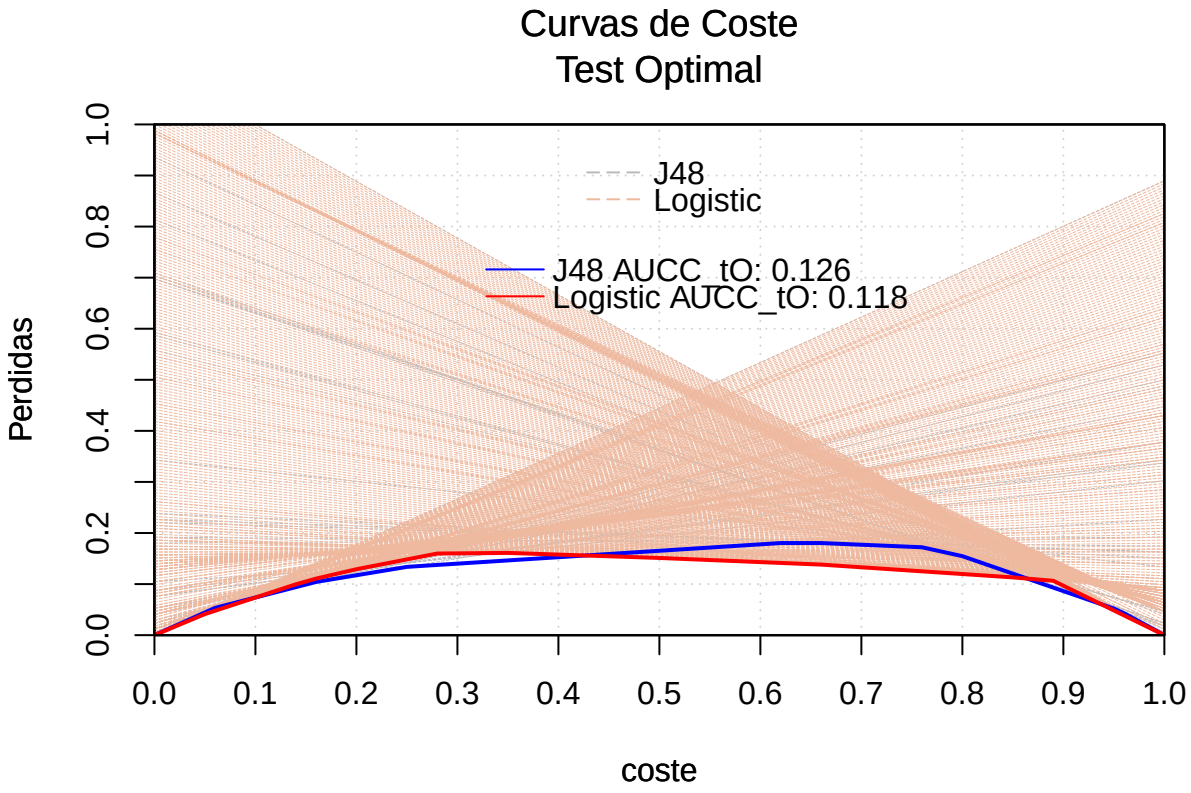


Comparación J48 vs. Regresión Logística

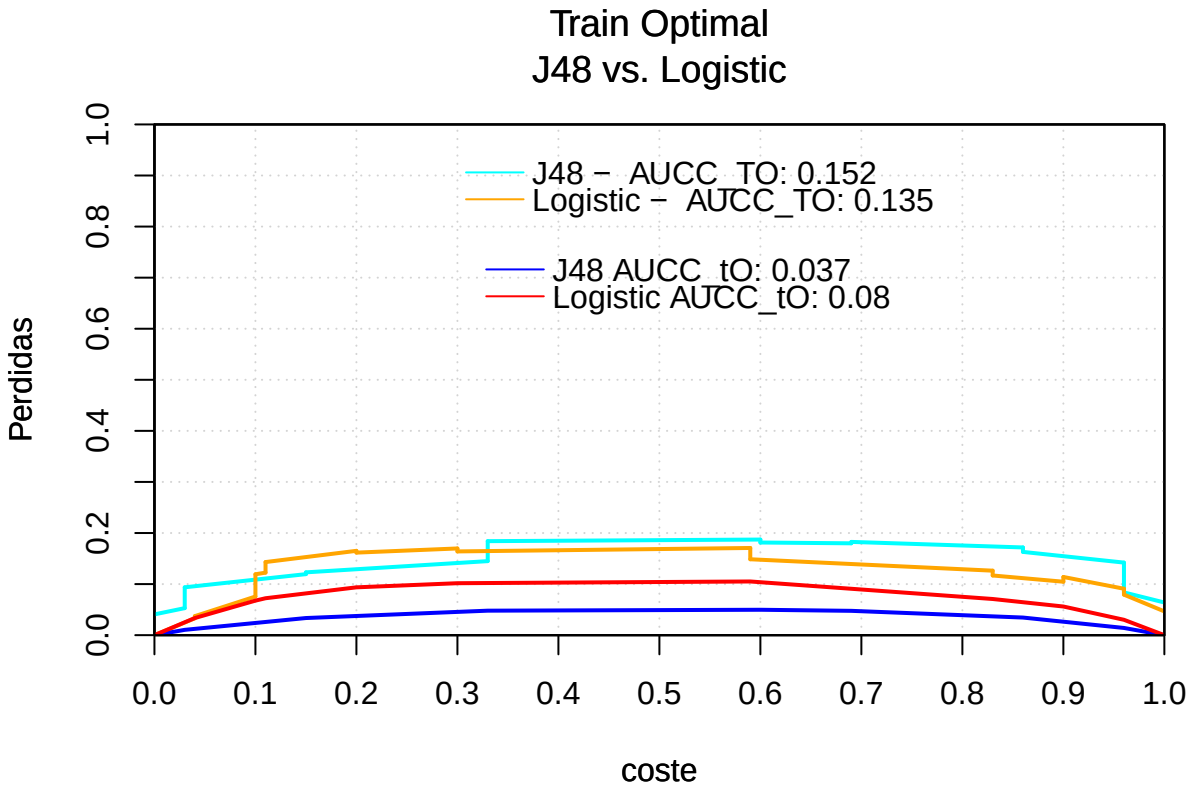
```
R <- RocCurves(list(probs_test, probs_test2), list(class_test, class_test2),
               plotCH = FALSE, main = "Curvas ROC", xlab = "TP",
               ylab = "FP", namesClassifiers = c("J48 - ","Logistic - "),
               xPosLegend = 0.35, yPosLegend = 0.7, cexL = 1)
```



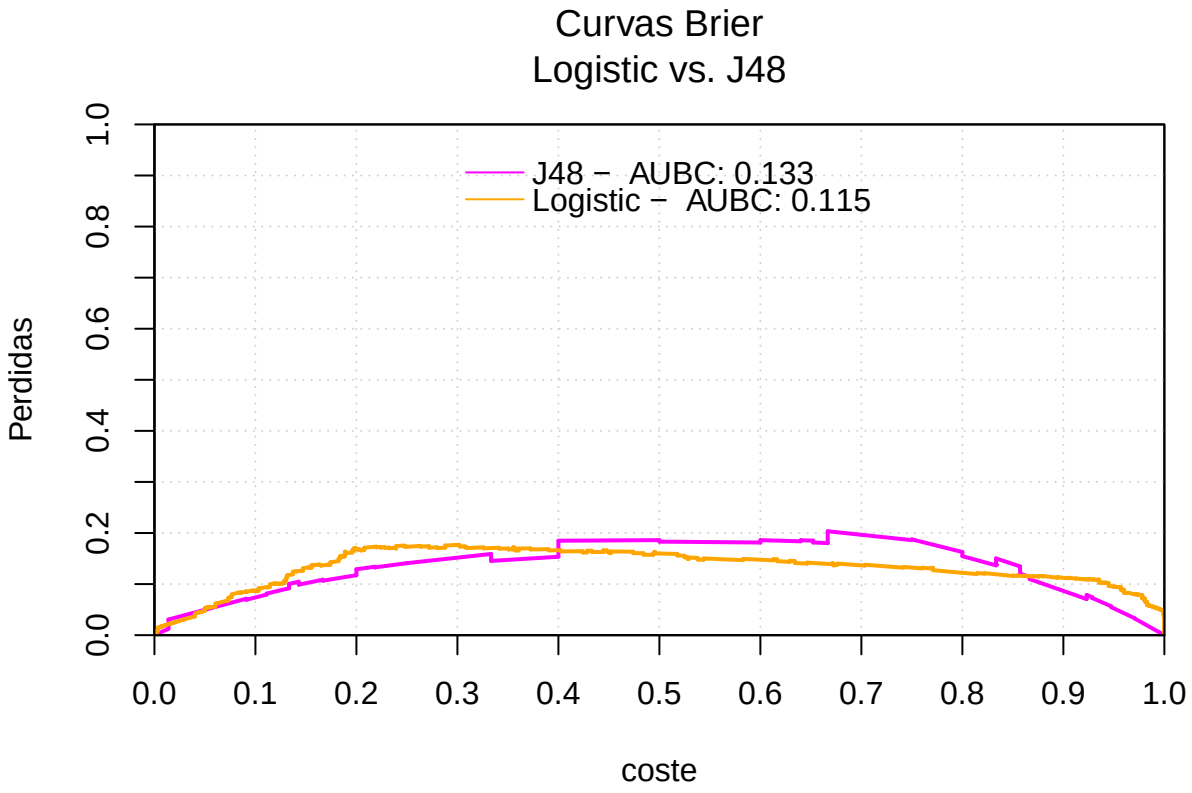

```
R <- CostLines(list(probs_test, probs_test2), list(class_test, class_test2),
               main = "Curvas de Coste\nTest Optimal", xlab = "coste",
               ylab = "Perdidas", namesClassifiers = c("J48","Logistic"),
               xPosLegend = 0.4, cexL = 1, lwd = 0.25, gridOFF = FALSE)

R <- TestOptimal(list(probs_test, probs_test2),
                 list(class_test, class_test2),
                 main = NULL, xlab = NULL,
                 ylab = NULL, namesClassifiers = c("J48","Logistic"),
                 xPosLegend = 0.3, yPosLegend = 0.78, cexL = 1, hold = TRUE)
```



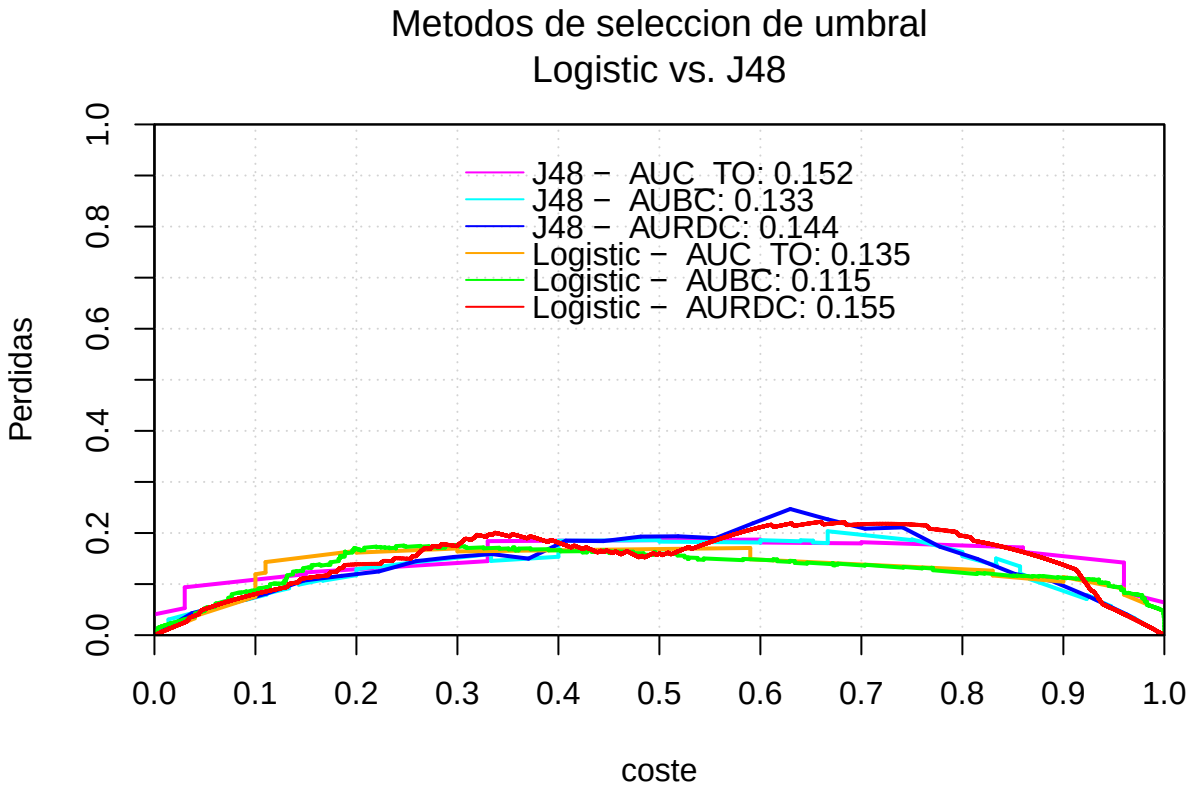

```
R <- TrainOptimal(predictions_test = list(probs_test, probs_test2),
                  classes_test = list(class_test, class_test2),
                  predictions_train = list(probs_train, probs_train2),
                  classes_train = list(class_train, class_train2),
                  main = "Train Optimal\nJ48 vs. Logistic", xlab = "coste",
                  ylab = "Perdidas", gridOFF = FALSE,
                  namesClassifiers = c("J48 - ", "Logistic - "),
                  xPosLegend = 0.28, cexL = 1, col=c("cyan", "orange"), lty=1)

 R <- TestOptimal(list(probs_train, probs_train2),
                 list(class_train, class_train2),
                 main = NULL, xlab = NULL,
                 ylab = NULL, namesClassifiers = c("J48","Logistic"),
                 xPosLegend = 0.3, yPosLegend = 0.78, cexL = 1, hold = TRUE)
```



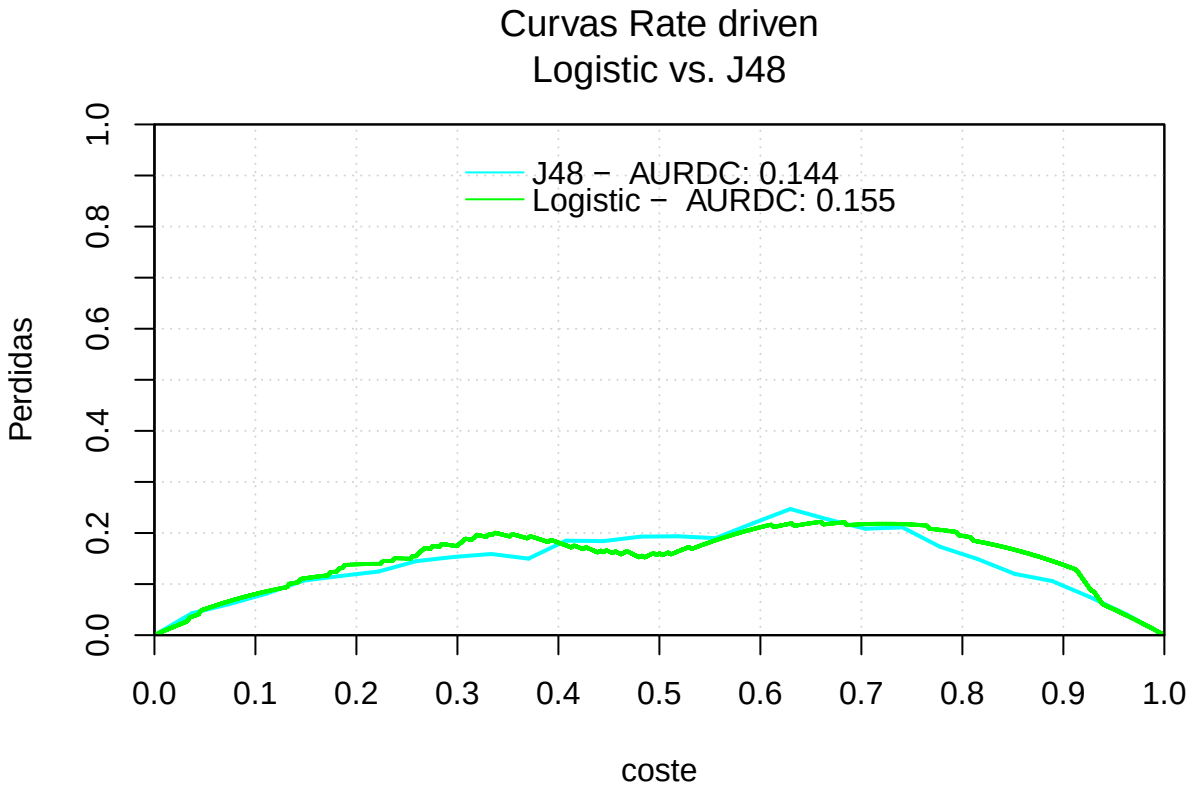

```
R <- CostCurves(list(probs_test, probs_test2),
                list(class_test, class_test2),
                main = "Curvas Brier\nLogistic vs. J48", xlab = "coste",
                ylab = "Perdidas",
                namesClassifiers = c("J48 - ","Logistic - "),
                cost_lines = FALSE, test_optimal = FALSE,
                score_driven = TRUE, xPosLegend = 0.28,
                col = list(c("magenta","orange")),
                lty=1, cexL = 1)
```




```
R <- CostCurves(list(probs_test, probs_test2),
                list(class_test, class_test2),
                main = "Metodos de seleccion de umbral\nLogistic vs. J48", xlab = "coste", predictions
                ylab = "Perdidas",
                namesClassifiers = c("J48 - ","Logistic - "),
                cost_lines = FALSE, test_optimal = FALSE, train_optimal = TRUE,
                score_driven = TRUE, xPosLegend = 0.28,
                rate_driven = TRUE,
                col = list(c("magenta","cyan", "blue"),c("orange", "green", "red")),
                lty=1, cexL = 1)
```




```
R <- CostCurves(list(probs_test, probs_test2),
                list(class_test, class_test2),
                main = "Curvas Rate driven\nLogistic vs. J48", xlab = "coste",
                ylab = "Perdidas",
                namesClassifiers = c("J48 - ","Logistic - "),
                cost_lines = FALSE, test_optimal = FALSE,
                rate_driven = TRUE, xPosLegend = 0.28,
                col = list(c("cyan", "green")),
                lty=1, cexL = 1)
```



Ejemplos Adicionales y uso de opciones gráficas de la librería

Carga de datos

```
data("predictions")
print(predictions)
```

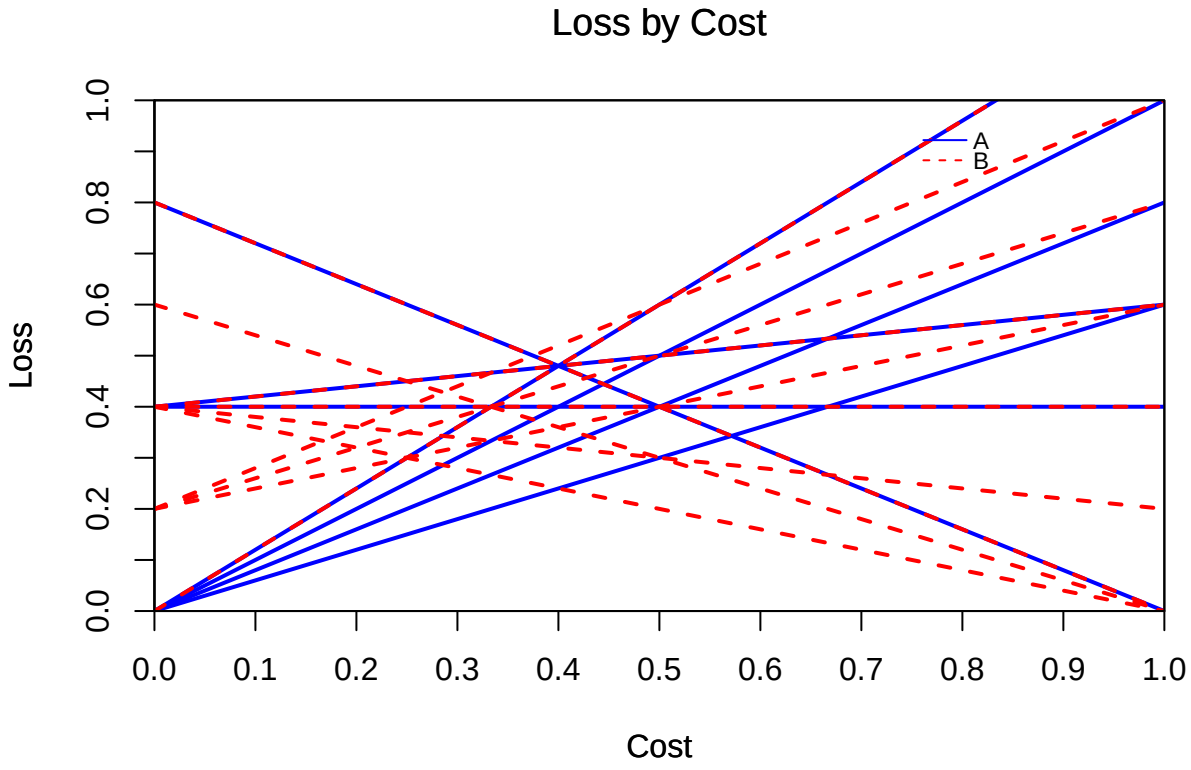
```
##    classes    A    B    C    D
## 1        1 0.70 0.60 0.00 0.65
## 2        1 0.80 1.00 1.00 0.90
## 3        1 0.80 0.95 0.93 0.88
## 4        1 0.70 0.25 0.91 0.48
## 5        0 0.80 0.68 0.78 0.74
## 6        0 0.75 0.64 0.83 0.70
## 7        0 0.10 0.37 0.78 0.24
## 8        0 0.55 0.30 0.95 0.43
## 9        0 0.80 0.72 1.00 0.76
## 10       0 0.15 0.25 0.87 0.20
```

```
classes<-predictions$classes
A<- predictions$A
B<- predictions$B
C<- predictions$B
```

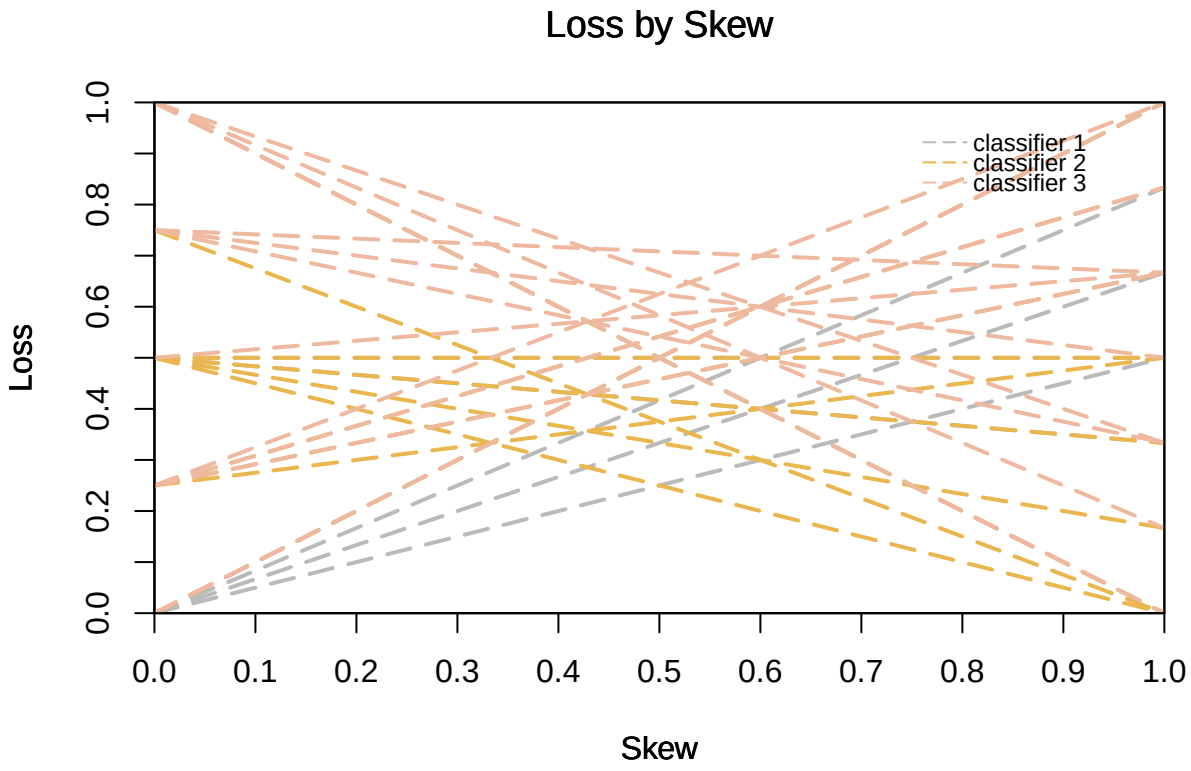
Examples

Cost Lines

```
#names legend
R <- CostLines(list(A, B), list(classes, classes),
               col=c("blue", "red"), lty=c(1, 2),
               namesClassifiers = c("A","B"))
```

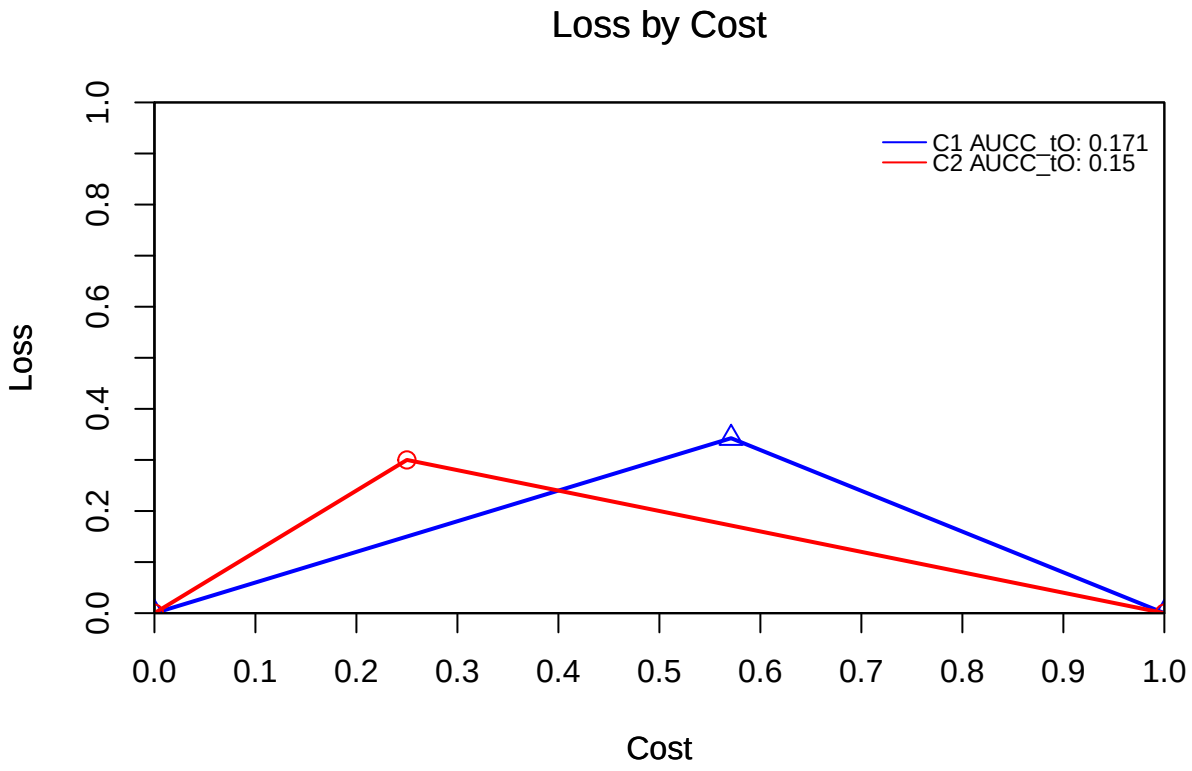



```
#Loss by cost and LegendOFF
R<- CostLines(list(A, B, rnorm(10)), list(classes),
              uniquec=TRUE, loss2skew = TRUE, lty=5)
```

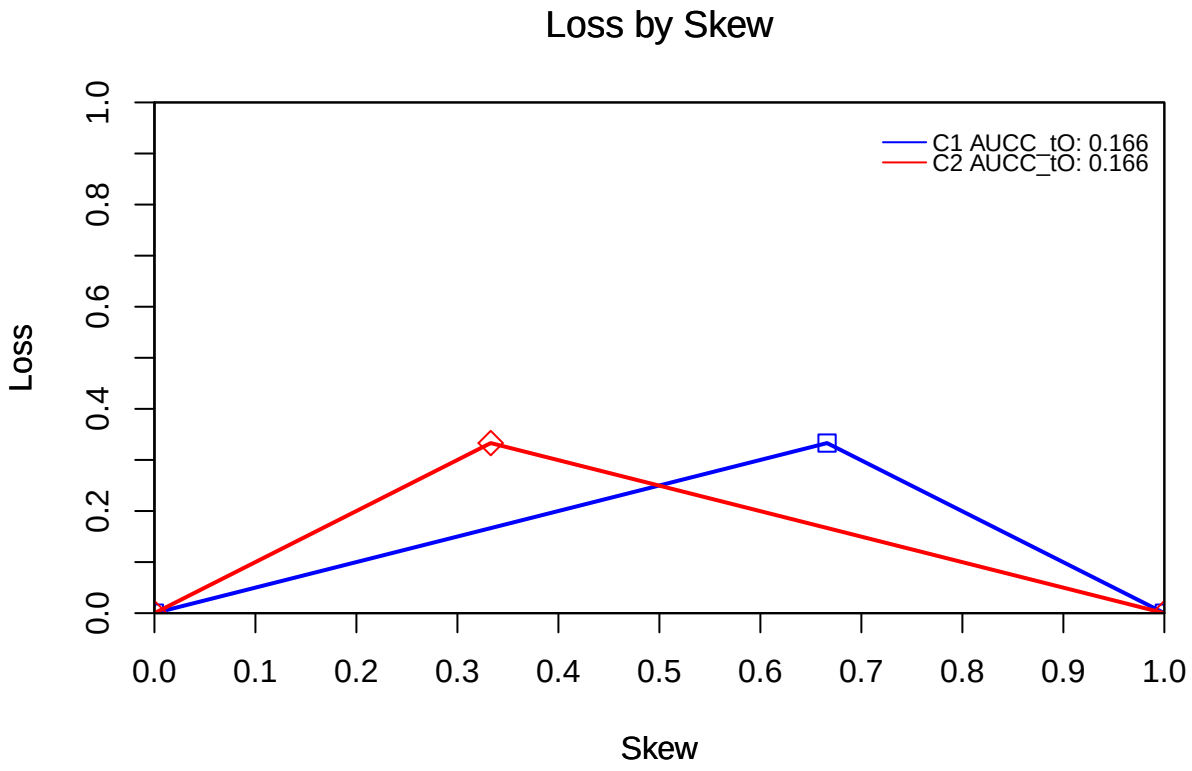


Test Optimal treshold choice method

```
#Loss by cost
R <- TestOptimal(list(A, B), list(classes),
                 uniquec=TRUE, pointsOFF = FALSE)
```

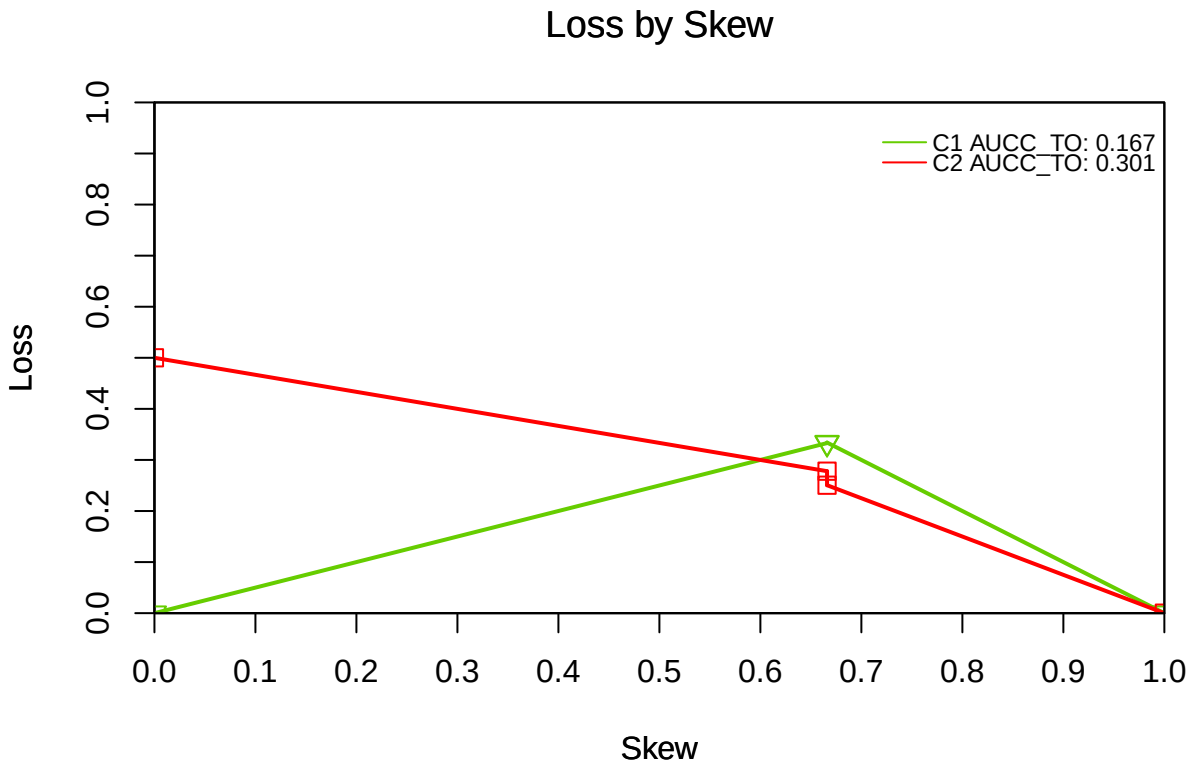



```
#Loss by skew
R <- TestOptimal(list(A, B), list(classes),
                 uniquec=TRUE,  loss2skew = TRUE,  pointsOFF=FALSE)
```

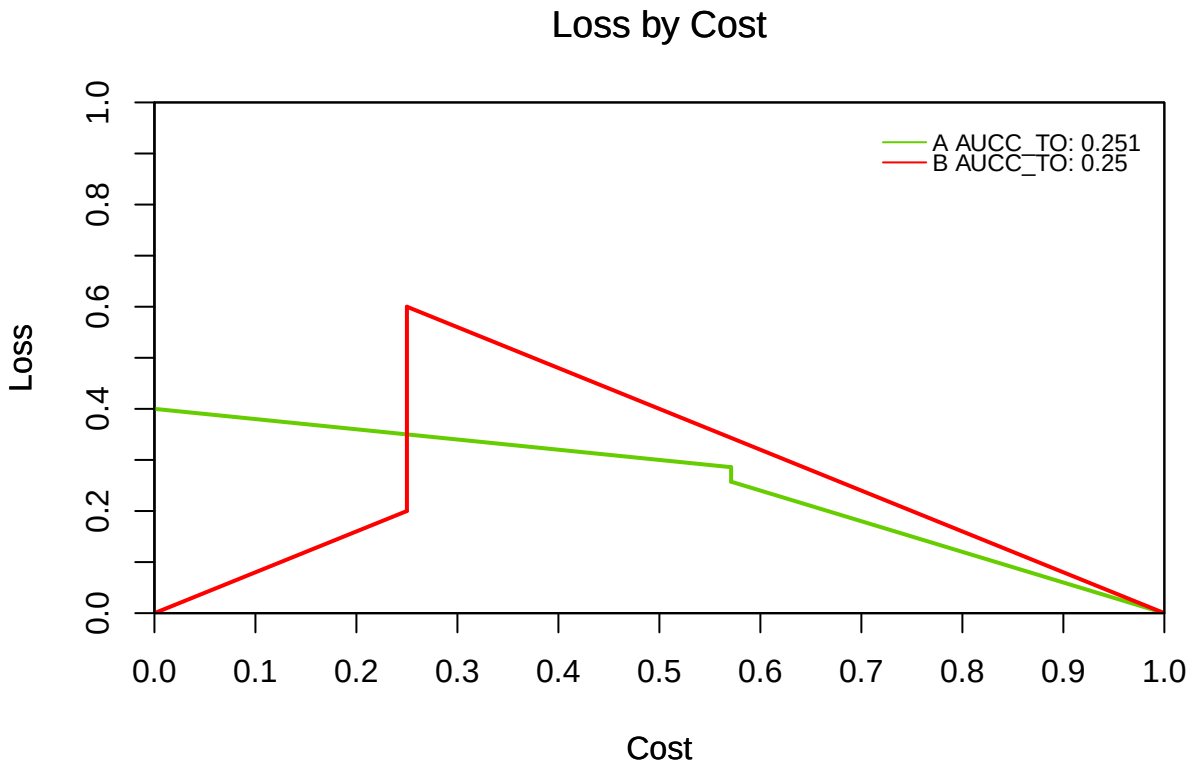


Train Optimal treshold choice method

```
#Loss by cost
R <- TrainOptimal(list(A, B), list(classes), list(A, B), list(classes),
                  pointsOFF=FALSE, uniquect = TRUE, uniquecT = TRUE,
                  loss2skew = TRUE, refuseT = TRUE)
```

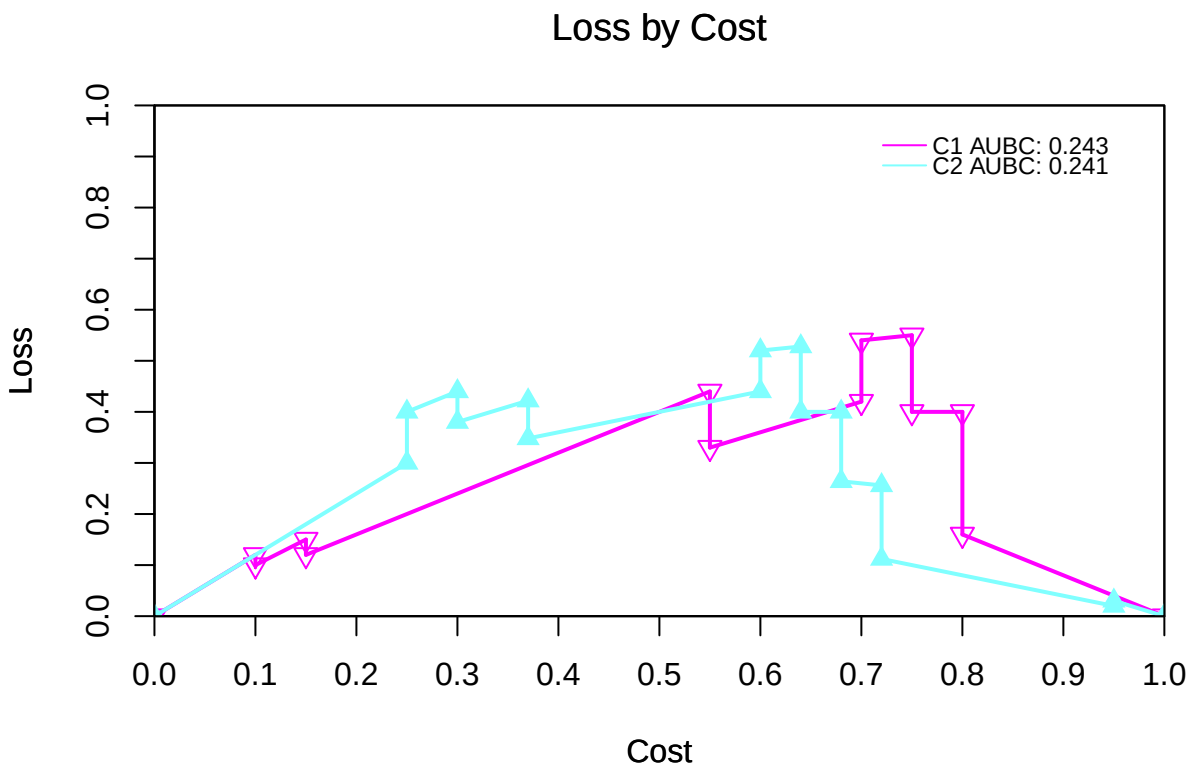



```
#Loss by skew
R <- TrainOptimal(list(A, B), list(classes),    list(B, A), list(classes),
                  uniquect = TRUE,  namesClassifiers=c("A", "B"),
                  namesTests=c("B","A"), uniquecT = TRUE)
```

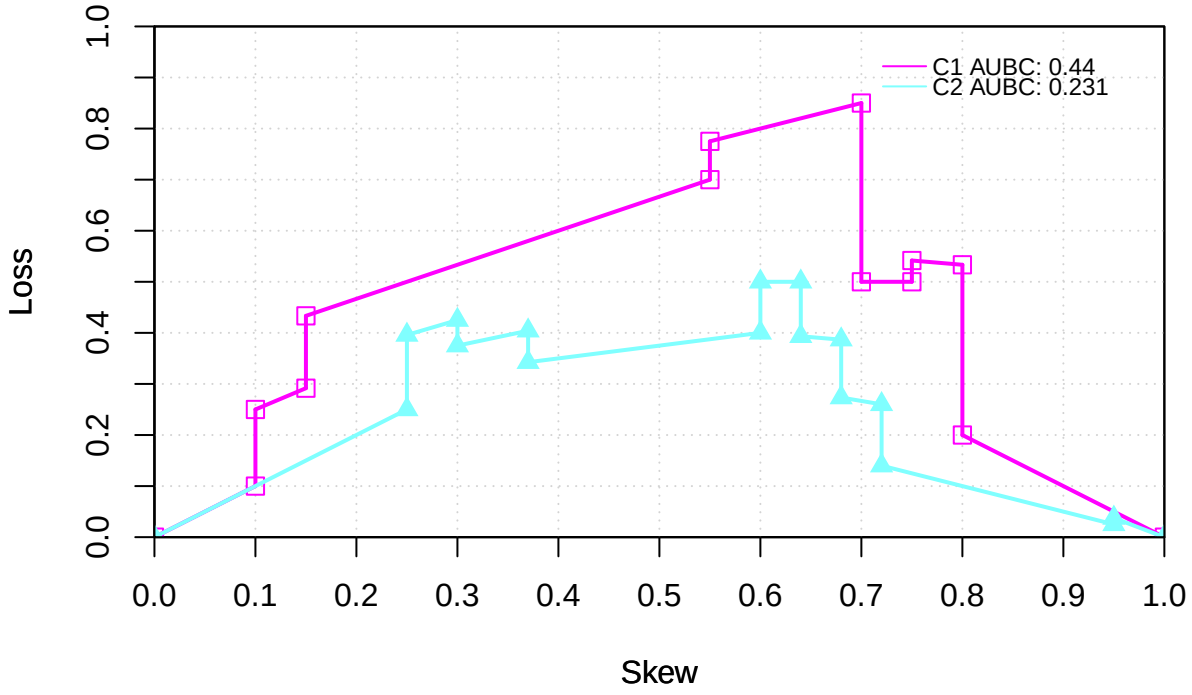


Brier Curves (score driven treshold choice method)

```
#Loss by cost
R <- BrierCurves(list(A, B), list(classes), uniquec = TRUE, pointsOFF=FALSE)
```

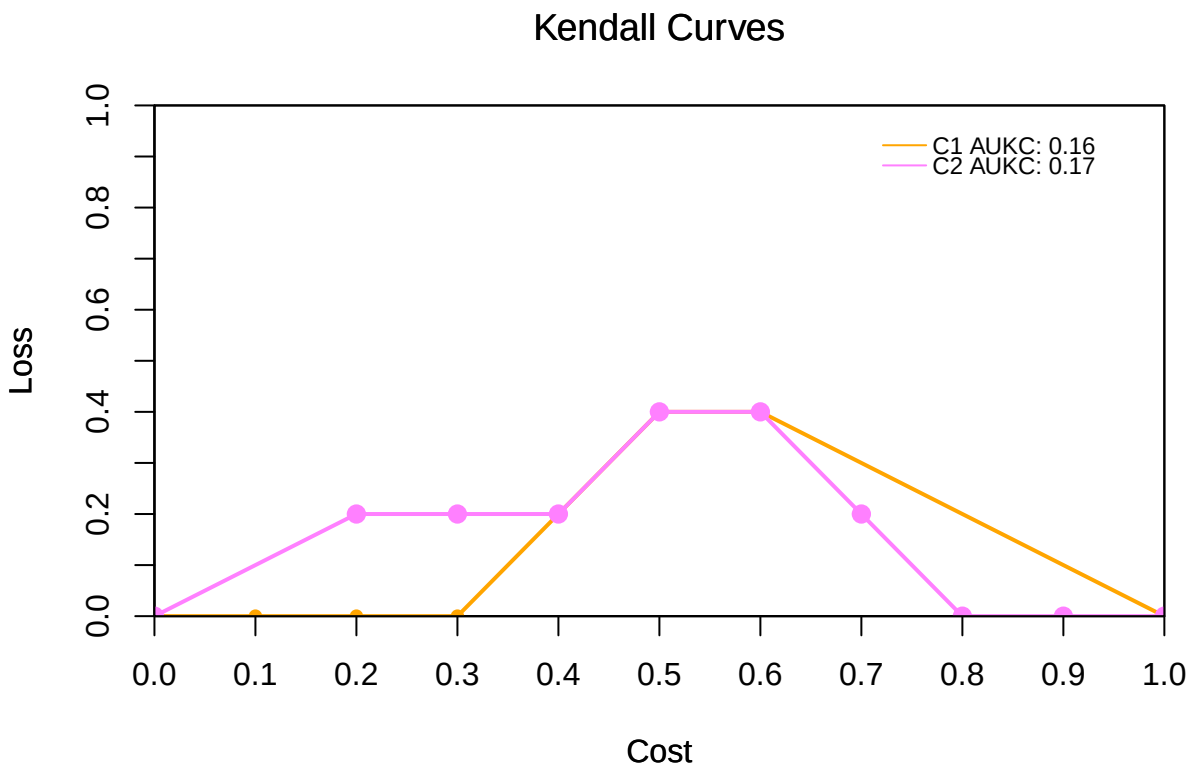



```
#Loss by skew
R <- BrierCurves(list(A, B), list((1-classes), classes), loss2skew = TRUE,
                 gridOFF = FALSE, pointsOFF=FALSE, main=NULL)
```

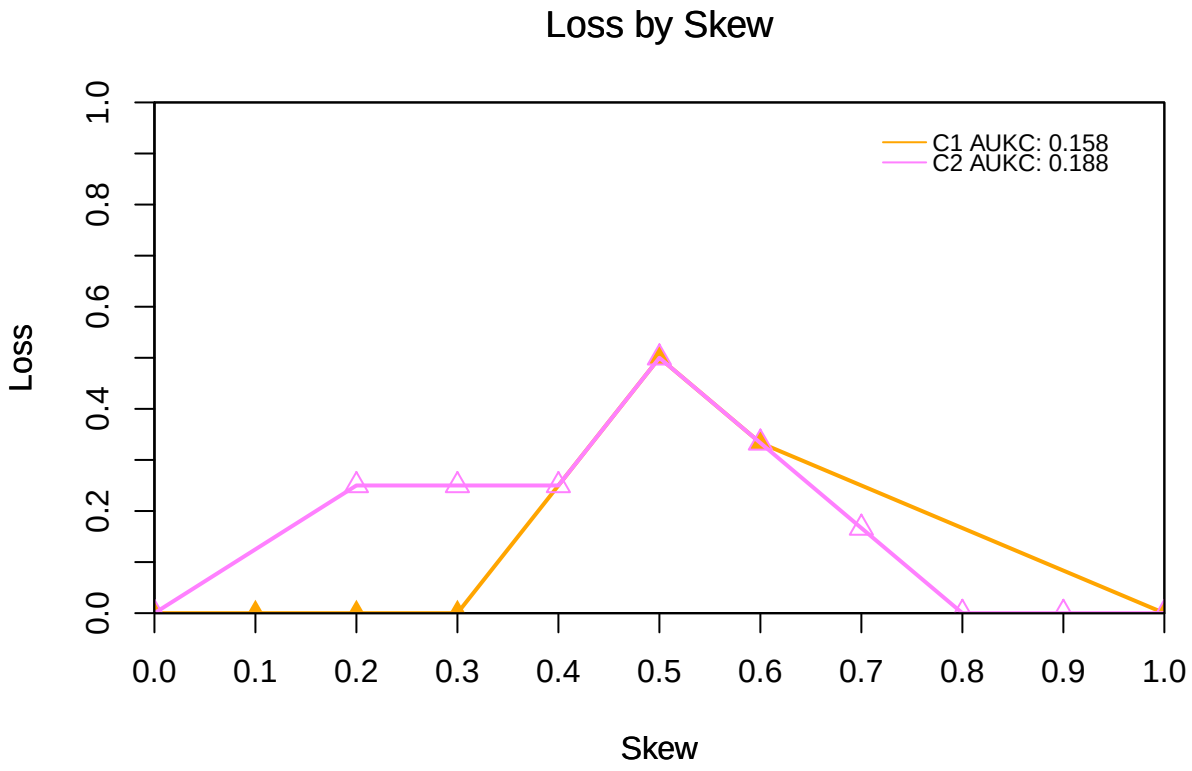


Kendall Curves (kendall treshold choice method)

```
#Loss by cost
R <- KendallCurves(list(A, B), list(classes), uniquec=TRUE,
                   pointsOFF=FALSE, main="Kendall Curves")
```

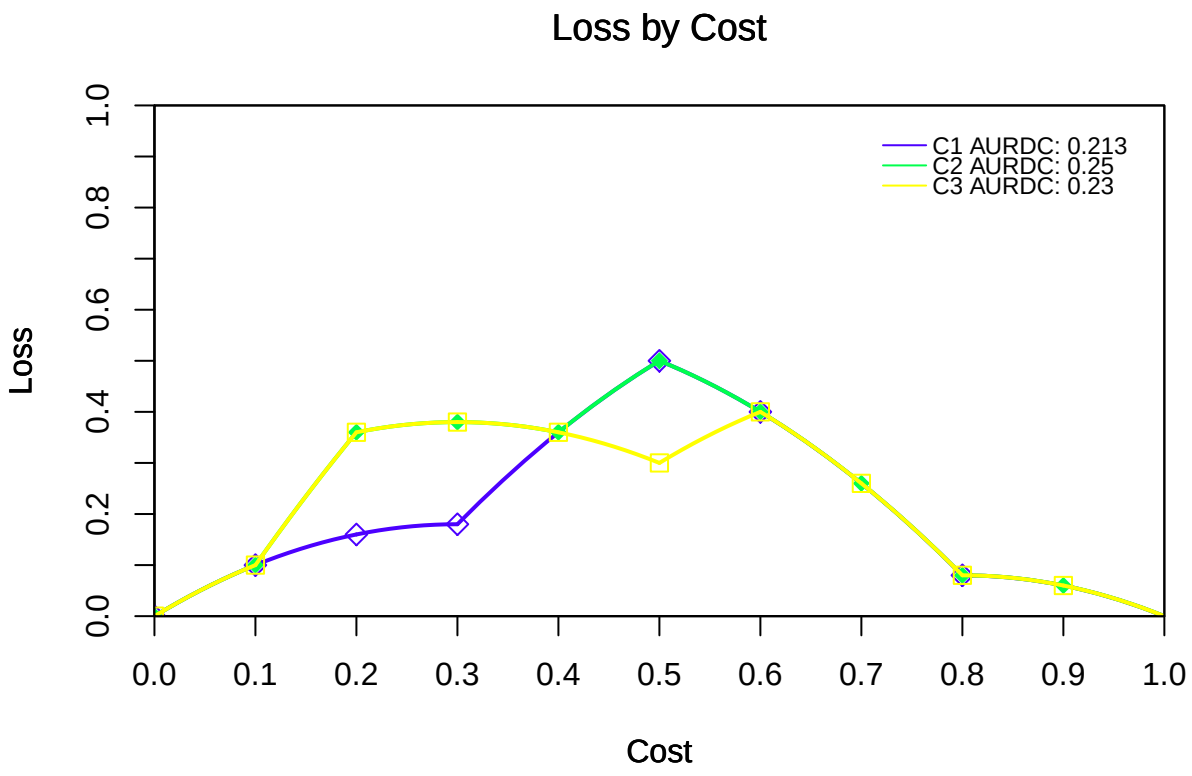



```
#Loss by skew
R <- KendallCurves(list(A, B), list(classes), uniquec=TRUE,
                   loss2skew = TRUE, pointsOFF=FALSE)
```

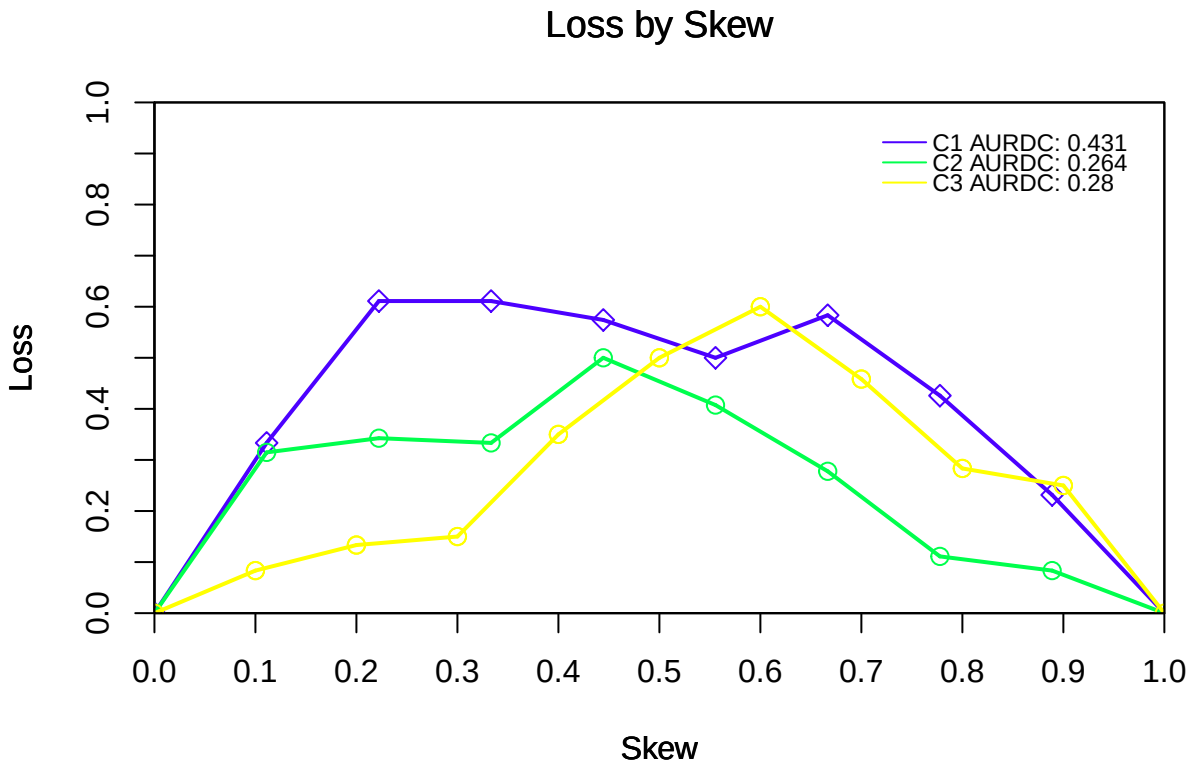


Rate Driven Curves (Rate Driven treshold choice method)

```
#Loss by cost
R<-RateDrivenCurves(list(A, B, rnorm(10)),list(classes),
                    pointsOFF=FALSE, uniquec=TRUE)
```

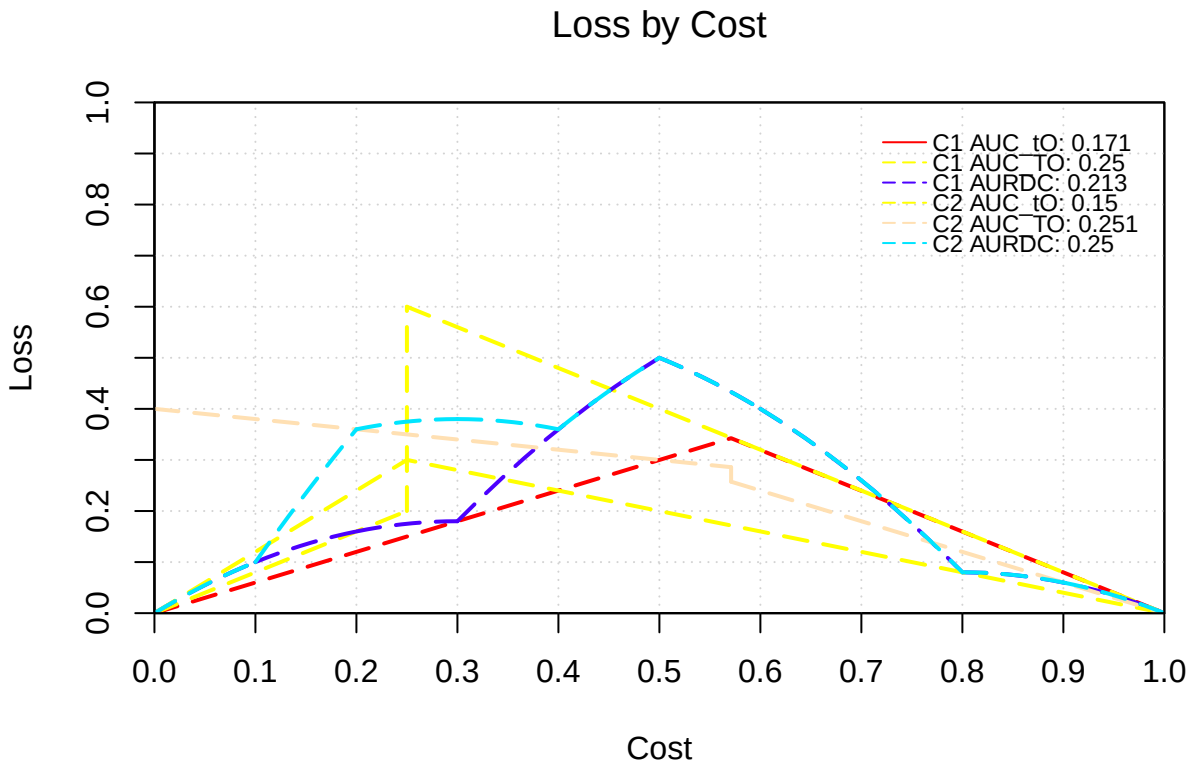



```
#Loss by skew
R<-RateDrivenCurves(list((1-C), B, rnorm(10)), list(classes),
                    pointsOFF=FALSE, uniquec=TRUE,  loss2skew = TRUE)
```

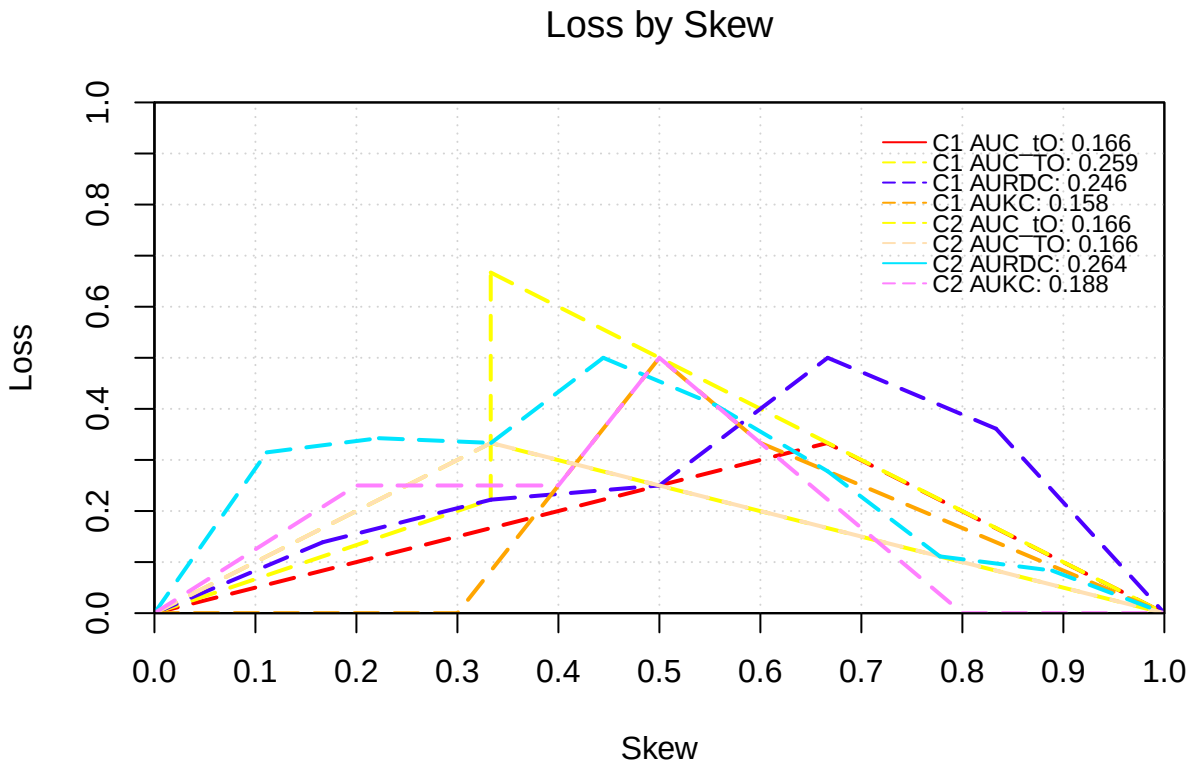


Full Options to plotting Cost Curves (CostCurves function)

```
R<-CostCurves(list(A, B), list(classes), cost_lines = FALSE, uniquec = TRUE,
              train_optimal = TRUE, predictionsT =  list(B, A),
              classesT = list(classes), uniquecT = TRUE, rate_driven = TRUE)
```

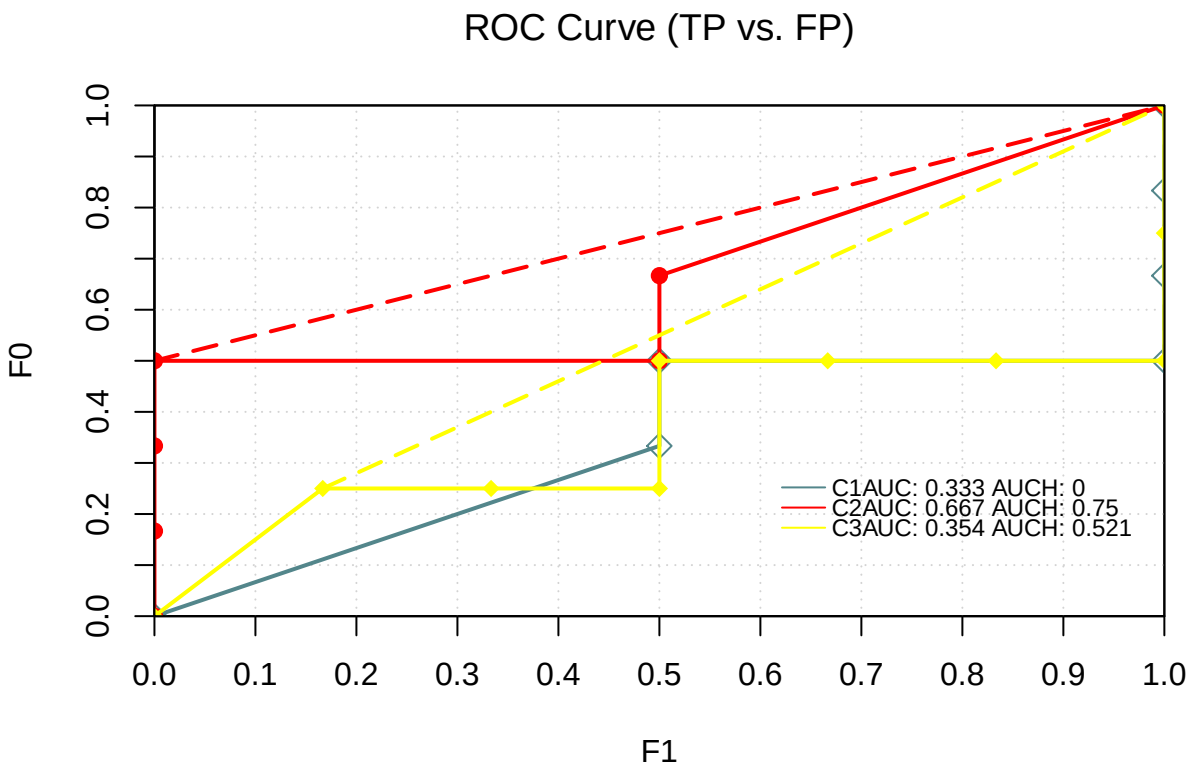



```
R<-CostCurves(list(A, B), list(classes), cost_lines = FALSE, loss2skew = TRUE,
              uniquec = TRUE, train_optimal = TRUE, predictionsT =  list(B),
              classesT = list(classes), uniqueTrain = TRUE,
              kendall_curves = TRUE, rate_driven = TRUE)
```

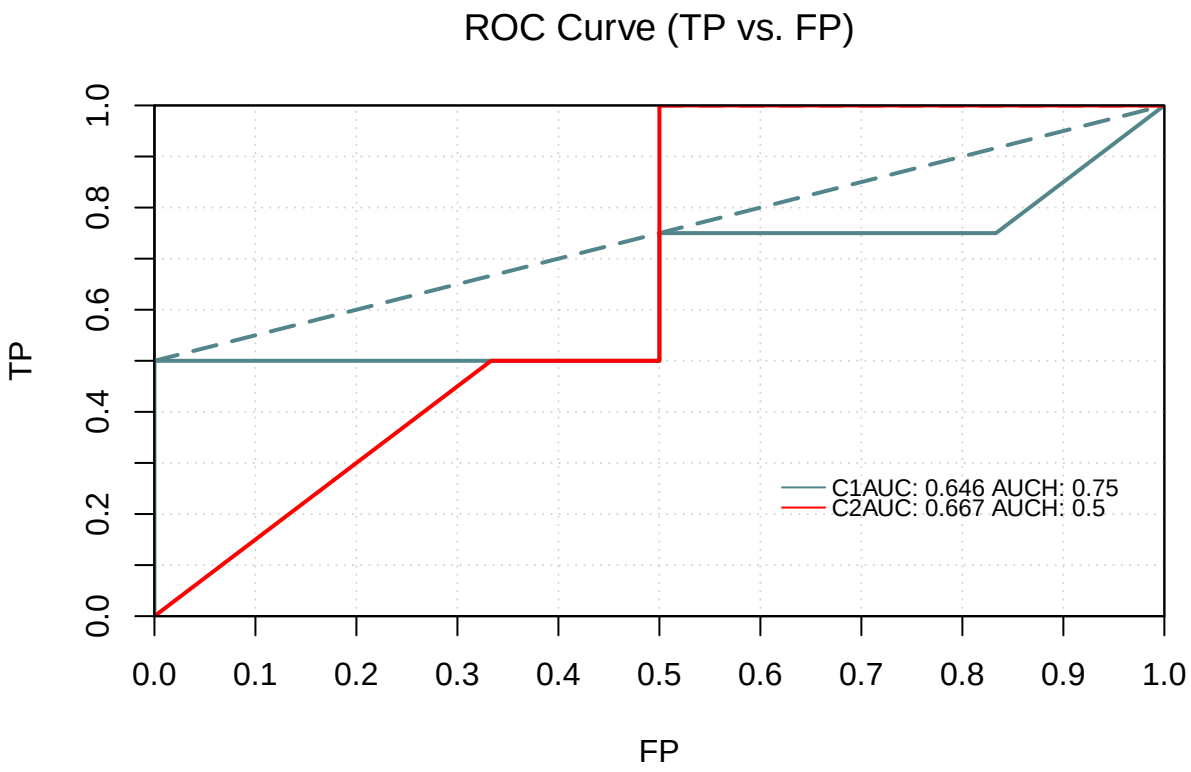


ROC Curves

```
#Example1
R<-RocCurves(list((1-A), A, B), list(classes, classes, (1-classes)),
             gridOFF=FALSE, pointsOFF=FALSE)
```




```
#Example2
R<-RocCurves(list(B, A), list(classes), gridOFF=FALSE,
             positiveY = TRUE, uniquec = TRUE, ylab="TP", xlab="FP")
```


ROC Curve (TP vs. FP)
TP
FP
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
C1AUC: 0.646 AUCH: 0.75
C2AUC: 0.667 AUCH: 0.5