UNIVERSIDAD
REY JUAN CARLOS

Máster en Data Science

Curso Académico 2019/2020

Trabajo Fin de Máster

OPTIMIZACIÓN Y AUTOMATIZACIÓN DE LA ELECCIÓN DE MODELOS DE CLASIFICACIÓN BINARIA SOBRE DIFERENTES TIPOS DE CONJUNTOS DE DATOS

Autor : Laura Martínez Borlaff

Tutor : José Felipe Ortega Soto

Trabajo Fin de Máster

Optimización y Automatización de la Elección de Modelos de Clasificación Binaria sobre Diferentes Tipos de Conjuntos de Datos

Autor : Laura Martínez Borlaff

Tutor : José Felipe Ortega Soto

La defensa del presente Trabajo Fin de Máster se realizó el día de de 2020, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Madrid, a de de 2020

Dedicado a

Adrián y Nico

II

Agradecimientos

Me gustaría agradecer este trabajo en primer lugar al cuerpo docente que formó parte del curso académico del máster, puesto que gracias a su tiempo y esfuerzo he aprendido y disfrutado de cada una de las clases.

También se lo quiero agradecer a mi marido, Adrián, por todo su apoyo durante el curso y después para que pudiera conseguir mi objetivo. No quiero tampoco olvidarme de mi hijo Nico, gracias por portarte bien y no dar mucho la lata a papá cuando tenía que ocuparse de ti para que pudiera enfocarme en este trabajo.

Resumen

En este trabajo de fin de máster se pretende mostrar como una herramienta automatizada para mostrar cuál es el modelo más óptimo en tareas de clasificación. Para realizar este trabajo se han utilizado diferentes tipos de conjuntos de datos: con pocas muestras, con muchas muestras, con datos numéricos, con datos categóricos, con datos mixtos (numéricos y categóricos), etc. para que los resultados puedan ser lo más generales posibles .

Para la realización de este trabajo se ha utilizado casi exclusivamente el paquete caret, además de otras librerías de apoyo. Los modelos seleccionados en este trabajo han constado de algunos vistos durante el curso y de otros nuevos, que se podrán ver en más detalle más adelante.

En la concepción de este trabajo, se pensó que sería una buena idea que fuera lo más automático posible, para que pudiera usarse como una herramienta de comparación de modelos. Este término de automático se refiere a que se ha intentado minimizar el número de intervenciones manuales del usuario. Esto quiere decir, que el usuario que pretenda usar esta herramienta, lo único que se le requiere es la creación de un fichero resumen (llamado metadata.txt) donde especifique las características del conjuntos de datos que quiera comparar, en posteriores secciones se darán más detalles sobre este paso.

El código realizado para este Trabajo de Fin de Máster puede consultarse en el siguiente enlace: `https://github.com/eleemebe/tfm/blob/master/main.R`

Índice general

Índice de figuras

Capítulo 1

Introducción

El desarrollo del poder de computación, la cantidad de datos de los que se disponen hoy en día y el bajo coste de almacenamiento en los últimos años han hecho resurgir técnicas de clasificación de datos que, habiendo sido estudiadas en el mundo académico a mediados del siglo XX, no habían sido aplicadas en la industria hasta la actualidad [5].

El área del "Aprendizaje automático" (en inglés, Machine Learning) es un aspecto cada vez más importante en los negocios y en la investigación actuales. Sus orígenes se pueden establecer en 1949 cuando Donald Hebb en su libro "The Organization of Behavior" [9] presenta teorías sobre estimulación y comunicación entre neuronas. Hebb escribió: "Cuando una célula ayuda repetidamente a disparar a otra, el axón de la primera célula desarrolla botones sinátipcos (alargándolos si ya existen) en contacto con el soma de la segunda célula". Esta teoría dio lugar a las bases de la inteligencia artificial y a las redes neuronales. Su modelo describe una forma de representar las relaciones entre neuronas artificiales (también llamadas nodos). La relación entre dos neuronas o nodos se fortalece si las dos neuronas se activan al mismo tiempo y se debilita si se activan por separado. La palabra "peso"se usa para describir estas relaciones, y se describe que los nodos o neuronas que tienden a ser tanto positivos como negativos tienen fuertes pesos positivos. Los nodos que tienden a tener pesos opuestos desarrollan fuertes pesos negativos [5].

En la década de 1950, Arthur Samuel de IBM desarrolló un programa de ordenador para jugar a las damas. Como el programa tenía una cantidad muy pequeña de memoria disponible, Samuel inició lo que se llama poda alfa-beta. Su diseño incluía una función de puntuación utilizando las posiciones de las piezas en el tablero. La función de puntuación intentó medir

las posibilidades de que cada lado ganase. El programa elige su próximo movimiento usando una estrategia minimax [10], que eventualmente evolucionó al algoritmo minimax. Samuel también diseñó una serie de mecanismos que permiten que su programa mejore. En lo que Samuel llamó aprendizaje de memoria, su programa registró todas las posiciones que ya había visto y lo combinó con los valores de la función de recompensa. A Arthur Samuel se le ocurrió por primera vez el concepto "Aprendizaje automático" en 1952 [5].

En 1957, Frank Rosenblatt, que trabajaba en el Laboratorio Aeronáutico de Cornell, combinó el modelo de interacción de las neuronas de Donald Hebb con los esfuerzos de aprendizaje automático de Arthur Samuel y creó el perceptrón [25]. El perceptrón. es un algoritmo para el aprendizaje supervisado de clasificadores binarios. Es un tipo de clasificador lineal, es decir, un algoritmo de clasificación que realiza sus predicciones en función de una función de predicción lineal que combina un conjunto de pesos con el vector de características. Aunque el perceptrón. parecía prometedor, no podía reconocer muchos tipos de patrones visuales (como las caras), lo que causaba frustración y detenía la investigación de las redes neuronales. Pasarían varios años antes de que las frustraciones de los inversores y las agencias de financiación desaparecieran. La investigación de redes neuronales o de aprendizaje automático tuvo problemas hasta su resurgimiento en la década de los 90 [5].

En 1967, se concibió el algoritmo conocido como Nearest Neighbor (en castellano, "vecino más cercano"), que fue el comienzo del reconocimiento básico de patrones. Este algoritmo se utilizó para mapear rutas y fue uno de los primeros algoritmos utilizados para encontrar una solución al problema del vendedor ambulante de encontrar la ruta más eficiente. A Marcello Pelillo se le ha dado el crédito por inventar la "regla del vecino más cercano" [4].

También en la década de 1960, el descubrimiento y uso de multicapas abrió un nuevo camino en la investigación de redes neuronales. Se descubrió que proporcionar y usar dos o más capas en el perceptrón ofrecía significativamente más potencia de procesamiento que un perceptrón que usa una única capa. Se crearon nuevas versiones de redes neuronales después de que se pasara del perceptrón a las "capas", y la variedad de redes neuronales continúa expandiéndose hoy en día. El uso de múltiples capas condujo a redes neuronales de retroalimentación y propagación hacia atrás (también llamada retropropagación). La retropropagación, desarrollada en la década de 1970, permite que una red ajuste sus capas ocultas de neuronas o nodos para adaptarse a nuevas situaciones. Describe 'la propagación hacia atrás de los errores", procesando

un error en la salida y luego distribuyéndolo hacia atrás a través de las capas de la red con fines de aprendizaje. La retropropagación ahora se está utilizando para entrenar redes neuronales profundas [5].

Una red neuronal artificial (Artificial Neural Network, ANN) tiene capas ocultas que se utilizan para modelar tareas más complicadas que los perceptrones anteriores. Las ANN son una de las herramientas principales utilizadas en el mundo del aprendizaje automático. Las redes neuronales usan capas de entrada y salida y, normalmente, incluyen una capa oculta (o varias capas ocultas) diseñada para transformar la entrada en datos que pueden ser utilizados por la capa de salida. Las capas ocultas son excelentes para encontrar patrones demasiado complejos para que un programador humano los detecte (patrones que un humano podría encontrar y luego codificarlos en un dispositivo) [5].

A fines de la década de 1970 y principios de la década de 1980, la investigación de Inteligencia Artificial (IA) se centraba cada vez más en utilizar enfoques lógicos basados en el conocimiento en lugar de algoritmos. Además, la investigación de redes neuronales fue abandonada por investigadores de ciencias de la computación e IA. Esto causó un cisma entre la inteligencia artificial y el aprendizaje automático, más centrado en algoritmos. Hasta entonces, Machine Learning se había utilizado como un programa de entrenamiento para IA. La industria del aprendizaje automático, que incluía una gran cantidad de investigadores y técnicos, se reorganizó en un campo separado y luchó durante casi una década. El objetivo de la industria pasó de la capacitación en Inteligencia Artificial a la resolución de problemas prácticos en términos de prestación de servicios. Su enfoque cambió de los enfoques heredados de la investigación de IA a los métodos y tácticas utilizados en la teoría de la probabilidad y las estadísticas. Durante este tiempo, la industria de Machine Learning mantuvo su enfoque en las redes neuronales y luego floreció en la década de 1990. La mayor parte de este éxito fue el resultado del crecimiento de Internet, que se benefició de la disponibilidad cada vez mayor de datos digitales y la capacidad de compartir sus servicios a través de Internet [5].

El “boosting”fue un desarrollo necesario para la evolución del aprendizaje automático. Los algoritmos de boosting se utilizan para reducir el sesgo durante el aprendizaje supervisado e incluyen algoritmos de aprendizaje automático que transforman a los aprendices débiles en fuertes. El concepto de boosting se presentó por primera vez en un artículo de 1990 titulado ”La fuerza de la capacidad de aprendizaje débil”(The strength of weak learnability), de Ro-

bert Schapire [22]. Schapire afirma: “Un conjunto de aprendices débiles puede crear un solo aprendiz fuerte”. Los aprendices débiles se definen como clasificadores que están ligeramente correlacionados con la clasificación verdadera (aún mejor que adivinar al azar). La mayoría de los algoritmos de boosting se componen de clasificadores débiles de aprendizaje repetitivo, que luego se suman a un clasificador fuerte final. Después de agregarlos, normalmente se ponderan de una manera que evalúa la precisión de los aprendices débiles. Luego, los pesos de los datos se ”vuelven a ponderar”. Los datos de entrada que se clasifican incorrectamente aumentan su peso, mientras que los datos clasificados correctamente pierden peso. Este entorno permite que los futuros aprendices débiles se centren más ampliamente en los aprendices débiles anteriores que fueron mal clasificados [5].

La diferencia básica entre los diversos tipos de algoritmos de boosting es ‘la técnicaütilizada para ponderar los puntos de datos de entrenamiento. AdaBoost es un algoritmo de aprendizaje automático popular e históricamente significativo, siendo el primer algoritmo capaz de trabajar con aprendices débiles (precisamente este es uno de los modelos recogidos en este trabajo de fin de máster que se verá más adelante). Algoritmos más recientes incluyen BrownBoost, LPBoost, MadaBoost, TotalBoost, xgboost y LogitBoost. Un gran número de algoritmos de boosting funcionan dentro del marco AnyBoost [5].

Machine Learning ha impulsado una nueva gama de conceptos y tecnologías, incluido el aprendizaje supervisado y no supervisado, nuevos algoritmos para robots, Internet de las cosas, herramientas de análisis, chatbots y más que son frecuentemente usadas hoy en día. Algunas de las formas más comunes donde actualmente se utiliza el aprendizaje automático en el mundo de los negocios son: análisis de datos de ventas, detección de fraude, recomendaciones de productos, precios dinámicos o procesamiento de lenguaje natural entre otros. Los algoritmos de Machine Learning combinados con las nuevas tecnologías informáticas promueven la escalabilidad y mejoran la eficiencia. Combinado con análisis de negocios, Machine Learning puede resolver una variedad de complejidades organizacionales. Los modelos modernos de Machine Learning se pueden utilizar para hacer predicciones que van desde brotes de enfermedades hasta el aumento y la disminución de las existencias [5].

1.1. Modelos de clasificación

En este proyecto nos centramos en modelos de clasificación. Un modelo de clasificación es una subcategoría del aprendizaje automático supervisado en la que el objetivo es predecir las etiquetas de clase categóricas de las nuevas instancias basándonos en observaciones pasadas. Estos modelos forman parte de los modelos de aprendizaje supervisado, los cuáles producen funciones a partir de un conjunto de ejemplos sobre los que conocemos la salida deseada [2]. Un modelo de clasificación binaria es aquel en el que las clases a predecir son dos, normalmente representadas con un 0 o un 1. Un caso típico sería predecir si un email es spam o no spam, o predecir si un cliente va a comprar un determinado producto (verdadero o falso). Los modelos de clasificación (binaria o no) se usan cada vez más en la industria en un gran abanico de aplicaciones, desde decidir si existe un gato en una imagen hasta determinar si hacer un descuento a un determinado cliente. Esto, a su vez, les hace aplicables a todo tipo de industrias, desde la bancaria al comercio online pasando por las relacionadas con el ámbito de la salud. Existen docenas de tipos de modelos capaces de generar predicciones de clasificaciones binarias (y continuarán descubriéndose más en el futuro), y sólo algunos de ellos se cubren en este trabajo de fin de máster. Muchos de estos modelos, a su vez, son en realidad un conjunto de modelos que dependiendo de unos parámetros de ajuste (hiperparámetros) producen predicciones diferentes.

Existe tal número de diferentes tipos de modelos de clasificación que no es sorprendente que en el ámbito de los modelos de clasificación binaria siempre le surjan dudas al analista o investigador a la hora de decidir qué modelo es el que mejor va a funcionar para el conjunto de datos sobre el que se va a trabajar. Probar todos los modelos con sus respectivos hiperparámetros para dar con el más adecuado es una tarea bastante ardua y muchas veces prohibitiva a nivel de costes (visión tiempo-beneficio). Esto crea inseguridad en los analistas o científicos de datos sobre si el modelo escogido para una aplicación es el mejor para el tipo de problema en concreto.

1.2. Objetivos del proyecto

Este Trabajo Fin de Máster intenta de alguna manera facilitar la decisión de qué tipo(s) de modelo probar dependiendo de la naturaleza de los datos de entrenamiento disponibles. En él, se evalúan diferentes tipos de datasets con diferentes tipos de modelos e hiperparámetros de una

manera consistente y objetiva para que así, cuando en el futuro se disponga de un dataset con unas características en concreto, se pueda tener información a priori sobre qué modelo y qué hiperparámetros han funcionado bien en tipos de datos parecidos, y por lo tanto poder fijar con más seguridad una primera línea de análisis para el conjunto de datos a analizar.

En este Trabajo de Fin de Máster también se evidencian diferencias entre lo que guía la teoría y lo que sucede en la práctica. Esto ocurre cuando los métodos de análisis son fundamentados en conjuntos de datos que no se encuentran tan fácilmente en la realidad o las propiedades utilizadas no son siempre las más correctas para cada problema de clasificación. Por ello algunos de los modelos utilizados, durante la realización de este trabajo, han destacado por no obtener un resultado como teóricamente se esperaba. Por ejemplo, en el caso de random forest, -normalmente- es un método de análisis que mejor funciona optimizando los recursos, pero como se verá más adelante en ciertos casos, y tras la realización de este trabajo, no sería el más indicado en ciertos tipos de datos, ya sea por el tiempo de ejecución o por la calidad de los resultados obtenidos.

Además, otro de los objetivos de este Trabajo de Fin de Máster es el de crear un herramienta que sea extensible para poder probar en el futuro nuevos datasets diferentes a los iniciales como también nuevos modelos, para así establecerse como una herramienta dinámica en la cual se apoyan los analistas o investigadores en los primeros pasos de su análisis.

Capítulo 2

Tecnologías y arquitectura

La arquitectura de este trabajo de fin de máster se puede describir con las fases típicas de un proyecto de Data Science: ingesta de datos de entrada, procesamiento/limpieza, entrenamiento, predicción, evaluación y visualización.

Para el desarrollo de estas fases se ha hecho uso principalmente del paquete caret disponible en el lenguaje de programación R (para las fases de entrenamiento y predicción), aunque también caben destacar las librerías ggplot2 para la fase de visualización y otras múltiples para las fases de ingesta, procesamiento y evaluación.

A continuación, en este apartado, explicaremos en detalle cómo hemos usado las librerías principales en cada una de las fases de la arquitectura de este trabajo. Además se añade al final de esta sección un esquema sobre la arquitectura seguida y el flujo de trabajo del programa escrito.

2.1. El paquete caret

Como se ha citado anteriormente en este trabajo se ha hecho uso del paquete caret de R. El paquete caret, cuyas siglas representan entrenamiento de clasificación y regresión (Classification And Regression Training en inglés) es un conjunto de funciones cuyo objetivo es intentar coordinar y simplificar el proceso de creación de modelos para problemas complejos de clasificación y regresión. Para ello el paquete cuenta con herramientas para [14] [12].:

- División del conjunto de datos.

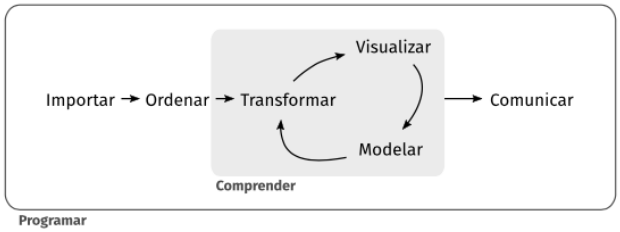

Figura 2.1: Ciclo de trabajo de projectos en ciencia de datos [7]

- Operaciones de preprocesado de los datos.
- Selección de características de los modelos a usar.
- Ajuste del modelo usando operaciones de remuestreo.
- Estimación de importancia de las variables de entrada.

El lenguaje de programación R ya cuenta con diferentes funciones de modelado, pero algunas de ellas tienen diferente sintaxis para cada modelo de entrenamiento y/o modelo de predicción. Por este motivo nació el paquete caret, se desarrolló como una manera para ofrecer una interfaz uniforme a las mismas funciones para estandarizar tareas comunes (por ejemplo en el ajuste de parámetros). El paquete caret se basa en multitud de paquetes de R pero intenta no cargarlos todos desde el inicio (por medio de la eliminación de las dependencias formales del paquete, consigue reducir considerablemente el tiempo de inicio del paquete) [12].

La instalación del paquete caret se hace por medio de la siguiente instrucción para asegurar que todos los paquetes necesarios están ya instalados [12]:

```
install.packages("caret",
          dependencies = c("Depends", "Suggests"))
```

```
Define sets of model parameter values to evaluate
for each parameter set do
    for each resampling iteration do
        Hold–out specific samples
        [Optional] Pre–process the data
        Fit the model on the remainder
        Predict the hold–out samples
    end
    Calculate the average performance across hold–out predictions
end
Determine the optimal parameter set
Fit the final model to all the training data using the optimal parameter set
```

Figura 2.2: Pseudocódigo función train de caret [12]

En el caso de que falte un paquete de modelado, aparecerá un mensaje de aviso para proceder a su instalación. La dependencia "Suggests" incluye 30 paquetes, que hemos instalado para una ejecución con menos incidencias. caret carga todos sus paquetes en el momento de ejecución, según sean necesarios.

Una de las herramientas principales en el paquete es la función `train` que se puede utilizar para [14]:

1. Evaluar, mediante remuestreo, el efecto de los parámetros de ajuste del modelo en el rendimiento [14],

2. Seleccionar el modelo óptimo en esos parámetros [14],

3. Estimar el rendimiento a partir del conjunto de entrenamiento [14].

Más concretamente ver figura 2.2.

La forma de uso típica de esta función es:

```
fit <- train(formula, data=train_data, method = model,
    trControl = fitControl, tuneGrid = tgrid)
```

`train` acepta un determinado conjunto de argumentos, de los cuáles sólo vamos a resaltar los utilizados en este trabajo de fin de máster.

Como podemos observar en la línea de código dispuesta arriba, la función `train` necesita como argumentos una expresión (fórmula) y los datos de entrada, en este caso los datos de entrenamiento o train, siguiendo una estructura similar a otras funciones propias de R. Caret permite indicar por un string el tipo de modelo de clasificación o regresión (regresión logística, random forest, etc) que se desee emplear por medio del argumento method. En este caso se pasa a la función `train` como método una lista con todos los modelos de clasificación elegidos en este trabajo para que los entrene uno a uno por medio de una iteración. Más concretamente esta lista cuenta con los siguientes modelos:

```
models <- list('rf', 'svmLinear', 'svmPoly', 'glm',
          'rpart', 'ada','xgbLinear',  'nb', 'knn')
```

Para cambiar los valores de los parámetros de ajuste, se pueden usar cualquiera de los argumentos tuneLength o tuneGrid. La función `train` puede generar un conjunto de posibles valores de parámetros y el argumento tuneLength controla cuántos se evalúan. Si queremos evaluar todos los enteros entre 1 y 15, deberemos establecer un tuneLength=15 para lograr esto. El argumento tuneGrid (empleado en este trabajo de fin de máster) se usa cuando se desean valores específicos. Se utiliza en un marco de datos donde cada fila es un ajuste de los parámetros de ajuste y cada columna se nombran igual que los parámetros de ajuste [14].

Continuando con la función `train`, el parámetro trControl debe tomar una lista de valores donde se especifica cómo debe actuar esta función por medio de la función `trainControl` [13]. `trainControl` se usa para modificar el método de remuestreo. La opción method controla el tipo de remuestreo a usar, donde por defecto, si no se especifica, toma el valor boot (bootstraping). En este caso se utilizó el método cv (acrónimo de Cross Validation, validación cruzada).

Hacemos un inciso para explicar en qué consiste la validación cruzada o Cross Validation. Normalmente en un proceso de aprendizaje automático, los datos se dividen en conjuntos de entrenamiento (también llamados train) y prueba (o test); el conjunto de entrenamiento se usa para entrenar el modelo y el conjunto de prueba se usa para evaluar el rendimiento de un modelo. Sin embargo, este enfoque puede conducir a problemas de sesgo y varianza (es bias and

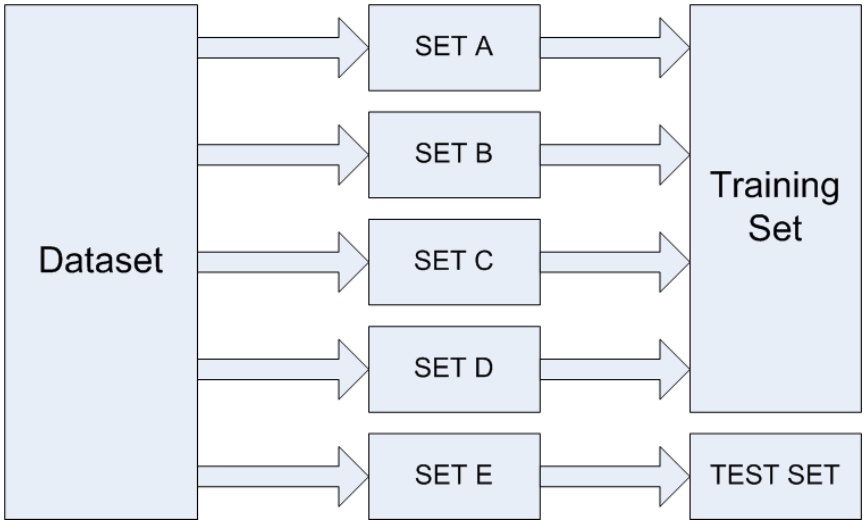

Figura 2.3: Esquema de funcionalidad de cross validation [18]

variance), en el que el modelo se ajuste demasiado a los datos de prueba. Explicado de otro modo, un problema de varianza se refiere al escenario en el que nuestra precisión obtenida en una prueba es muy diferente a la precisión obtenida en otro conjunto de tests usando el mismo algoritmo [11] [18].

Una solución a este problema es usar K-Fold Cross-Validation para la evaluación del rendimiento donde K es cualquier número. El proceso de validación cruzada de particiones en K es conceptualmente sencillo. Se dividen los datos en K particiones. Fuera de las particiones K, los conjuntos K-1 se usan para entrenamiento mientras que el conjunto restante se usa para pruebas. El algoritmo se entrena y se prueba K veces, cada vez que se usa un nuevo conjunto como conjunto de prueba, mientras que los conjuntos restantes se usan para el entrenamiento. Finalmente, el resultado de la validación cruzada K-Fold es el promedio de los resultados obtenidos en cada conjunto. Supongamos que queremos realizar una validación cruzada de 5 veces (K=5). Para hacerlo, los datos se dividen en 5 conjuntos, por ejemplo, los denominamos SET A, SET B, SET C, SET D y SET E. El algoritmo se entrena y se prueba K veces. En el primer conjunto de datos, SET A a SET D se usan como conjunto de entrenamiento y SET E se usa como conjunto de prueba como se muestra en la figura 2.3 [18].

En el segundo conjunto de datos, SET A, SET B, SET C y SET E se usan para el entre-

namiento y SET D se usa como prueba. El proceso continúa hasta que cada conjunto se use al menos una vez para entrenamiento y una vez para pruebas. El resultado final es el promedio de los resultados obtenidos con todos las particiones. De esta manera podemos reducir el efecto de la varianza. Usando la desviación estándar de los resultados obtenidos de cada partición podemos de hecho encontrar la varianza en el resultado general [18].

El parámetro K de CrossValidation utilizado en este trabajo se incorpora en el parámetro number. Este parámetro especifica la validación cruzada en un determinado número de K-fold, que especifica el número de repeticiones; de no pasarse un valor predeterminado su valor por defecto es 10.

```
fitControl <- trainControl(method = "cv", number = 2,
           allowParallel = T, classProbs = TRUE)
```

El argumento classProbs no es más que un lógico (True or False) el cual permite (valor True) o no (valor False) calcular las probabilidades de clase para los modelos de clasificación (junto con la clase pronosticada) en cada muestra. Además mediante el argumento classProbs igualado a True permitimos el uso de un backend paralelo que esté cargado y disponible [12].

Por defecto, R no aprovechará todos los núcleos disponibles en un ordenador. A fin de ejecutar código en paralelo, primero se debe hacer que el número deseado de núcleos esté disponible para que R pueda registrar un ”backend paralelo”, que crea efectivamente un clúster en el que los cálculos pueden ser enviados. Independientemente de la aplicación, la computación en paralelo se reduce a tres pasos básicos: dividir el problema en partes, ejecutar en paralelo y recopilar los resultados [16]. Mediante el parámetro **allowParallel** lo que indicamos es que en el caso de que exista un backend en paralelo cargado y listo para su uso, la función deberá utilizarlo o no. En nuestro caso sí nos interesaba esta funcionalidad para optimizar el tiempo de ejecución y por ello hemos añadido este parámetro a nuestra función.

Una vez entrenados los datos, pasamos a la fase de predicción. Para predecir nuevas muestras utilizamos la función **predict.train**.

```
predictions <- predict.train(fit, newdata=test_data, type = "prob")
```

Esta función en los modelos de clasificación sigue un comportamiento predeterminado, el cual calcula la clase predicha. La opción **type** igualada a “prob” se puede usar para calcular las

Model	Method Value	Type	Libraries	Tuning Parameters
Random Forest	rf	Classification, Regression	randomForest	mtry
Support Vector Machines with Linear Kernel	svmLinear	Classification, Regression	kernlab	C
Support Vector Machines with Polynomial Kernel	svmPoly	Classification, Regression	kernlab	Degree, scale, C
Generalized Linear Model	glm	Classification, Regression		None
CART	rpart	Classification, Regression	rpart	cp
Boosted Classification Trees	ada	Classification	Ada, plyr	iter, maxdepth, nu
eXtreme Gradient Boosting	xgbLinear	Classification, Regression	xgbboost	Nrounds, lambda, alpha, eta
Naive Bayes	nb	Classification	klaR	fL, usekernel, adjust
k-Nearest Neighbors	knn	Classification, Regression		k

Cuadro 2.1: Modelos recogidos en este trabajo.

probabilidades de clase del modelo (interesante en nuestro caso para representar la curva ROC). Si esta opción estuviera igualada a “raw” devolvería en su lugar un vector con las clases de la predicción.

La mayoría de los modelos de clasificación en R producen tanto una predicción de clase como las probabilidades para cada clase. Para datos binarios, habitualmente, la predicción de clase se basa en un límite de probabilidad del 50 %. En este trabajo de fin de máster, al intentar ser agnóstico al dataset de entrada, se ha mantenido el umbral al 50 %. No obstante, cabe resaltar que en áreas en las que un falso positivo sea más costoso que un falso negativo, se podría considerar establecer un umbral más elevado y al contrario, en áreas en las que un falso negativo sea más costoso que un falso positivo se podría considerar poner un umbral más bajo.

Con caret, el uso de predict (objecto, newdata) devuelve la clase predicha y predict (object, new data, type = ‘prob”) devuelve probabilidades específicas de la clase (cuando el objeto es generado por la función `train`).

Los modelos de caret que han sido usados como objeto de estudio en este trabajo han sido los siguientes: rf, svmLinear, svmPoly, glm, rpart, ada, xgbLinear, nb, knn. Todos estos modelos están disponibles en el paquete caret para la función `train`. El código detrás de todos los protocolos que se pueden usar puede ser obtenido usando la función getModelInfo. En la tabla 2.1 se resume de forma esquemática cada uno de los detalles de los parámetros de entrenamiento, sólo se han seleccionado los que se han utilizado en este trabajo. La mayoría de los modelos elegidos pueden usarse tanto para clasificación o regresión, solo Boosted Classification Trees (ada) y Naive Bayes (nb) son exclusivos para problemas de clasificación [15]. Ver tabla 2.1

Para cada uno de los parámetros de cada modelo se ha creado una lista de valores sobre los que se entrena el modelo. En el apartado sobre resultados se verá cuál funciona mejor en cada modelo dependiendo de qué tipo de datos se disponga.

2.2. Otras librerías usadas

Además del paquete caret se han utilizado otras librerías para la realización de este trabajo. Como bien se ha comentado anteriormente, este trabajo se ha basado sobretodo en el uso de la librería caret, pero se han necesitado de otras librerías para diferentes fases en su realización.

2.2.1. Readxl

Una de las primeras librerías que necesitan ser cargadas para al ejecución del nuestro script es Readxl. El paquete readxl facilita la extracción de datos de Excel y de R. Comparado con muchos otros de los paquetes existentes (por ejemplo, gdata, xlsx, xlsReadWrite), readxl no tiene dependencias externas, por lo que es fácil de instalar y usar en todos los sistemas operativos. Está diseñado para trabajar con datos tabulares. Este paquete admite tanto el formato .xls heredado como el formato .xlsx moderno basado en xml. La biblioteca libxls C se utiliza para admitir .xls, que abstrae muchas de las complejidades del formato binario subyacente (para información sobre su instalación, ir al enlace de referencia) [29].

2.2.2. Stringr

Las cadenas de caracteres o strings juegan un papel importante en muchas tareas de limpieza y preparación de datos. Para ello, el paquete stringr [30] proporciona un conjunto coherente de funciones diseñadas para que trabajar con cadenas sea lo más fácil posible. El paquete de stringr está construido sobre stringi, que utiliza la biblioteca ICU C para proporcionar implementaciones rápidas y correctas de manipulaciones de cadenas comunes. stringr se centra en las funciones de manipulación de cadenas más importantes y comúnmente utilizadas, mientras que stringi proporciona un conjunto completo que cubre casi todo lo que pueda imaginar. Hay cuatro familias principales de funciones en stringr [30]:

1. Manipulación de caracteres: que permiten manipular caracteres individuales dentro de las cadenas en vectores de caracteres.

2. Herramientas para agregar, eliminar y manipular espacios en blanco.

3. Operaciones locales cuyas operaciones variarán de un lugar a otro.

4. Funciones de coincidencia de patrones. Estos reconocen cuatro motores de descripción de patrones. El más común son las expresiones regulares, pero hay otras tres herramientas.

En general, las principales diferencias entre el lenguaje R base y el stringr son [30]:

- Las funciones stringr comienzan con el prefijo str_. Las funciones de cadena base R no tienen un esquema de nomenclatura consistente.

- El orden de las entradas suele ser diferente entre R base y el stringr. En R base, el patrón a coincidir generalmente viene primero; en stringr, la cadena a manipular siempre viene primero. Esto hace que stringr sea más fácil de usar con lapply() o purrr :: map().

- La salida y entrada de funciones stringr ha sido cuidadosamente diseñada. Por ejemplo, la salida de str_locate() se puede alimentar directamente a str_sub(). Esto no es ocurre para funciones como regpexpr() y substr().

- Las funciones básicas de R usan argumentos (como perl, fixed e ignore.case) para controlar cómo se interpreta el patrón. Para evitar la dependencia entre argumentos, stringr utiliza funciones auxiliares (como fixed(), regexp() y coll ()).

2.2.3. ROC

Otra de las librerías empleadas en este trabajo ha sido el paquete ROC [23]. Este paquete es un paquete de R para evaluar y visualizar el rendimiento del clasificador. Las curvas de sensibilidad/especificidad, los gráficos de elevación y los gráficos de precision/recall son ejemplos populares de visualizaciones de compensación para pares específicos de medidas de rendimiento. ROCR es una herramienta flexible para crear curvas de rendimiento 2D con parámetros de corte combinando libremente dos de más de 25 medidas de rendimiento (se pueden agregar nuevas medidas de rendimiento utilizando una interfaz estándar). Las curvas de diferentes ejecuciones

de validación cruzada o bootstrapping se pueden promediar mediante diferentes métodos, y las desviaciones estándar, los errores estándar o los gráficos de caja se pueden usar para visualizar la variabilidad entre las ejecuciones. La parametrización se puede visualizar imprimiendo valores de corte en las posiciones de curva correspondientes o coloreando la curva de acuerdo con el corte. Todos los componentes de una gráfica de rendimiento se pueden ajustar rápidamente utilizando un mecanismo flexible de envío de parámetros. Además de su flexibilidad ya que se integra perfectamente con las instalaciones de gráficas integradas en R, ROCR es fácil de usar, con solo tres comandos y valores predeterminados razonables para todos los parámetros opcionales.

En este TFM se pensó en utilizar el área debajo de la ROC (AUROC) para comparar los diferentes modelos pero finalmente se optó por el F1 score, por lo que el paquete ROC se usó solamente para análisis, quedando fuera de la implementación final.

2.2.4. PRROC

Este paquete calcula las áreas bajo la curva de precisión/recall (PR) y las características operativas del receptor (ROC) para datos ponderados y no ponderados. La curva PR y ROC son medidas valiosas del rendimiento de un clasificador. A diferencia de los paquetes R disponibles, PRROC [6] permite calcular las curvas PR y ROC y las áreas debajo de estas curvas para datos etiquetados de manera suave utilizando una interpolación continua entre los puntos de las curvas PR. La interpolación entre puntos de la curva PR se realiza mediante una función por partes no lineal. Además, PRROC proporciona una función de trazado genérico para generar gráficos con calidad de publicación de curvas PR y ROC. Además de las áreas debajo de las curvas, las curvas mismas también se pueden calcular y trazar mediante un método S3 específico.

Al igual que el paquete ROC, el paquete PRROC quedó fuera de la implementación final ya que se pensó en utilizar el área debajo de la PRROC para comparar los diferentes modelos pero finalmente se optó por el F1 score, por lo que el paquete PRROC se usó solamente para análisis.

2.2.5. Ggplot2

La visualización de datos es una técnica utilizada para la representación gráfica de datos [20]. Al usar elementos como diagramas de dispersión, cuadros, gráficos, histogramas, mapas,

etc., hacemos que nuestros datos sean más comprensibles. La visualización de datos facilita el reconocimiento de patrones, tendencias y excepciones en nuestros datos. Nos permite transmitir información y resultados de una manera rápida y visual.

Es más fácil para un cerebro humano comprender y retener información cuando se representa en forma pictórica. Por lo tanto, la visualización de datos nos ayuda a interpretar los datos rápidamente, examinar diferentes variables para ver sus efectos en los patrones y obtener información de nuestros datos.

En este trabajo hemos utilizado ggplot2 para la representación de los resultados obtenidos. ggplot2 es un sistema para crear gráficos declarativamente, basado en The Grammar of Graphics. En este paquete el usuario solo se tiene que preocupar casi de proporcionar los datos, y decir a ggplot2 cómo asignar variables a la estética, qué primitivas gráficas usar, y ésta se ocupa de los detalles [28].

2.2.6. gridExtra

Esta librería proporciona una serie de funciones de nivel de usuario para trabajar con gráficos de çuadrícula", en particular para organizar múltiples diagramas basados en cuadrículas en una página y dibujar tablas [1].

Es decir este paquete resulta muy útil y sencillo para ensamblar varios gráficos en una página, para ello cuenta con la función grid.arrange(). Con grid.arrange(), se puede reproducir el comportamiento de las funciones base par(mfrow = c (r, c)), especificando el número de filas o columnas.

Si se omiten por completo los parámetros de diseño, grid.arrange() calculará un número predeterminado de filas y columnas para organizar los gráficos.

Se pueden lograr diseños más complejos pasando dimensiones específicas (anchos o alturas) o una matriz de diseño que defina la posición de cada parcela en una cuadrícula rectangular.

2.3. Automatización de la ingesta, proceso de datos, entrenamiento y evaluación

Uno de los objetivos de este trabajo fin de máster era el de crear una herramienta que fuera lo más automatizada posible y extensible en un futuro a nuevos modelos y tipos de datos. Para ello, el código realiza todo el proceso desde la lectura de los diferentes conjuntos de datos, la limpieza de datos -de esto se hablará más adelante-, el entrenamiento de los modelos, su evaluación y la creación de visualizaciones en una ejecución de manera automatizada, es decir, sin intervención manual. Tan solo existe un paso manual, la categorización de los datasets para su lectura.

2.3.1. Categorización de datasets (creación de metadata.txt)

Para la correcta interpretación de las variables de entrada por los diferentes modelos, es necesario saber si la variable es de naturaleza categórica o numérica, o si es por el contrario la variable a predecir (target). Esto no se puede inferir desde el valor de la variable, dado que por ejemplo, variables que parecen tomar valores numéricos (por ejemplo una que represente el año) son de naturaleza categórica.

Para pasar esa información a los diferentes modelos del trabajo, se ha definido un fichero ”metadata.txt” que acompaña a cada dataset (dentro del directorio de entrada DATASETS, que contiene un subdirectorio separado para cada dataset de la UCI elegido), en el que se detalla explícitamente esta y otra información. El fichero metadata.txt tiene el siguiente formato:

```
data: <nombre del fichero de datos>
column[1-N]: <tipo de columna> (target, categorical o numerical)
format: <formato del fichero para su correcta lectura>
              (xlsx, comma, semicolon, tab, space)
```

2.3.2. Lectura de datos

En los directorios para cada uno de los datasets, se encuentran dos ficheros: un fichero que contiene los datos (con formatos .csv, .txt, xlsx, etc) y un metadata.txt. El fichero de datos es el fichero original que se descargó del repositorio UCI, sin ningún tipo de manipulación. El fichero

metadata.txt es un fichero creado a mano explicado en la sección anterior, cuyo propósito es exclusivamente el de recoger las características o la información sobre el fichero de datos. El código realizado lee este fichero para recoger la información sobre cómo leer y adecuar/limpiar los datos para su posterior uso. Este fichero indica el nombre del fichero de datos que describe, tipo de datos contenidos en cada columna del fichero de datos, si contiene variables numéricas, categóricas, etc.; y cuál es la columna a predecir (o target). Además indica el formato que tiene el fichero de datos si es separado por comas, tabuladores, espacios, etc. Así el código recibe ligeras indicaciones a priori sobre cómo insertar los datos.

2.3.3. Limpieza/preprocesado

Las operaciones que realiza el código para limpiar los datos son bastante estándar, ya que deben ser compatibles con todos los datasets y todos los modelos. Por ello, una vez se han leído los datos del fichero que los contiene se realizan el siguiente procesado:

- las variables de entrada se convierten en valores numéricos o categóricos, según se indique el archivo de metadata de cada dataset (los modelos en caret usan de forma diferente variables categóricas (factors) y numéricas).
- se eliminan los ejemplos que tienen valores no disponibles (N/As)
- se eliminan de los datos de entrenamiento aquellas variables categóricas que disponen de un único valor, al carecer de información

2.3.4. Separación en grupos de train y test

Cada uno de los ficheros de datos, los dividimos en dos conjuntos train y test (entrenamiento y evaluación). Del conjunto de test extraemos la columna target y creamos un fichero CSV con dicha información. El directorio llamado ”actuals” contiene las columnas target de los datos de test.

2.3.5. Entrenamiento y estimación de hiperparámetros

Se entrenan todos cada uno de los datasets con cada uno de los modelos elegidos probando diferentes valores para cada hiperparámetro de cada modelo con una estrategia de Cross

Validation para que los parámetros generalicen a datos no vistos.

2.3.6. Evaluación

Una vez entrenados los modelos, se usan para predecir los datos de test. Todos los ficheros de salida son guardados en un directorio llamado output. El resultado de cada modelo se guarda en un fichero CSV que recibe un nombre combinado con el nombre del dataset y el nombre del modelo entrenado, para su mejor identificación.

Finalmente comparamos el resultado con los ficheros del directorio ”actuals”los cuáles contienen la información correcta a predecir, y calculamos las métricas del rendimiento de cada uno de los modelos.

2.4. Esquema de la arquitectura construida

En la figura 2.4 adjunta se resume brevemente el flow del código propuesto. En la primera imagen, los cuadros grandes en blanco ocuparía todos los procesos internos que hace el código realizado. En la imagen posterior se puede observar el detalle esquematizado de cada uno de los procesos que realiza, nombrando a la función en cada paso y su respectiva entrada y salida.

En algunas de las fases o pasos se utilizan más funciones de las que se citan en el esquema, obviamente se han resumido con fines aclaratorios, para simplificar el esquema de lo que realiza el código propuesto.

Como nota, recordar que los ficheros metadatas tienen que ser creados a mano antes de la ejecución del código, puesto que este fichero es el que aporta detalles al código sobre cómo manejar los datos a introducir.

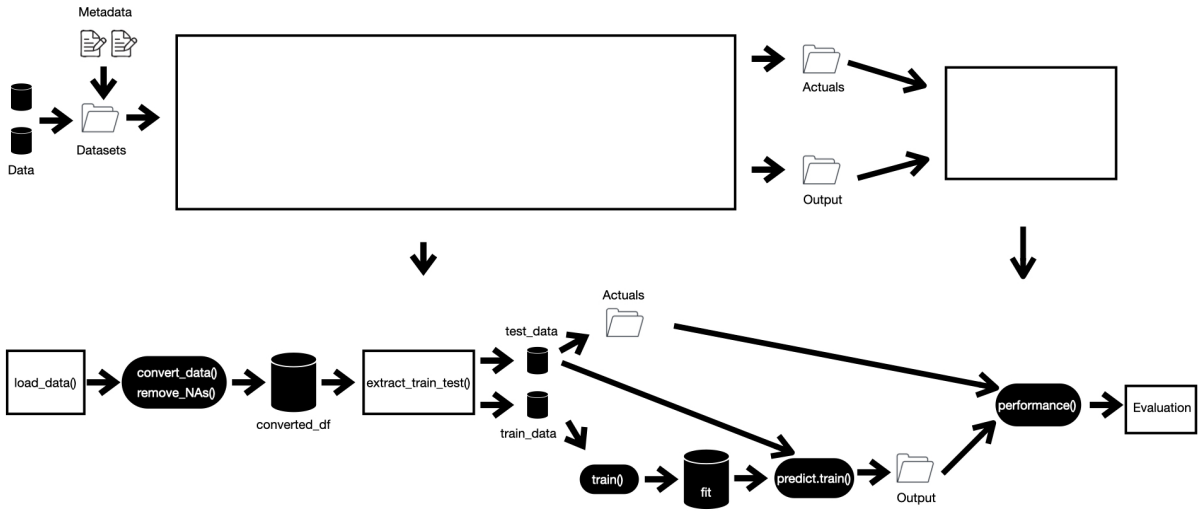

Figura 2.4: Esquema de funcionalidad del código

Capítulo 3

Diseño experimental

En apartados anteriores se han explicado las librerías usadas, funciones codificadas, datos y modelos utilizados, y demás de manera individual. En este apartado se explicará el diseño experimental, dando una visión más cohesiva del proyecto, desde el por qué se han escogidos los modelos escogidos, pasando por qué conjuntos de datos han sido escogidos y por qué, hasta cómo se ha decidido comparar los modelos, entre otros aspectos.

3.1. Conjuntos de datos usados

Empezaremos por los datos escogidos. La idea de este trabajo de fin de máster se centraba en algoritmos de clasificación desde un principio, por ello los conjuntos de datos deberían ser de específicos para tareas de clasificación. Todos los conjuntos de datos utilizados en la realización de este trabajo de fin de máster han sido obtenidos del repositorio online para machine learning de la Universidad Irvine de California (UCI) [27]. La búsqueda de datos de para este tipo de tareas fue bastante fácil puesto que, gracias a que en el repositorio de UCI, se pueden filtrar directamente todos los datasets dependiendo del tipo de tarea que se vaya a realizar. Luego a la hora de elegir entre todos los datasets que aparecen se hizo especial hincapié en que cumplieran una serie de requisitos:

- Que la tarea por defecto fuera de clasificación
- Que la variable a predecir fuera **binaria**

Name	Data Types	Default Task	Attribute Types	Number Instances	Number Variables	Year	Binary Classification?	File name
Acute Inflammations	Multivariate	Classification	Categorical, Integer	120	6	2009	yes	acute
Balloons	Multivariate	Classification	Categorical	16	4		yes	balloons
banknote authentication	Multivariate	Classification	Real	1372	5	2013	yes	banknote
BLOGGER	Multivariate	Classification	N/A	100	6	2013	yes	blogger
Breast Cancer	Multivariate	Classification	Categorical	286	9	1988	yes	breast-cancer
Cryotherapy Dataset Data Set	Univariate	Classification	Integer, Real	90	7	2018	yes	cryotherapy
Diabetic Retinopathy Debrecen Data Set	Multivariate	Classification	Integer, Real	1151	20	2014	yes	messidor_features
Shuttle Landing Control	Multivariate	Classification	Categorical	15	6	1988	yes	shuttle-landing-control
Spambase	Multivariate	Classification	Integer, Real	4601	57	1999	yes	spambase
SPECT Heart	Multivariate	Classification	Categorical	267	44	2001	yes	SPECT
SPECTF Heart	Multivariate	Classification	Integer	267	44	2001	yes	SPECTF
Teaching Assistant Evaluation	Multivariate	Classification	Categorical, Integer	151	5	1997	yes	tae.txt
Thoracic Surgery Data	Multivariate	Classification	Integer, Real	470	17	2013	yes	thoracicSurgery.txt
Tic-Tac-Toe Endgame	Multivariate	Classification	Categorical	958	9	1991	yes	tic-tact-toe.data.txt
Blood Transfusion Service Center	Multivariate	Classification	Real	748	5	2008	yes	transfusion
Breast Cancer Wisconsin (Diagnostic)	Multivariate	Classification	Real	569	32	1995	yes	wdbc
Wholesale customers	Multivariate	Classification, Clustering	Integer	440	8	2014	yes	wholesame customers data.csv

Cuadro 3.1: Conjuntos de datos utilizados en este trabajo

- Que los variables fueran categóricos o numéricos (reales, enteros, etc). Datasets con variables de tipo texto no fueron incluidos en este trabajo.

Otros aspectos como el número de variables o muestras no fueron sujeto de elección, puesto que se intentaba tener un ejemplo de datasets diferentes entre sí. Además se intentó también elegir datasets con diferentes órdenes de magnitud en lo que se refiere al número de muestras para que el trabajo fuera lo más generalista posible.

El UCI Machine Learning Repository es una colección de bases de datos, teorías de dominio y generadores de datos que son utilizados por la comunidad de Machine Learning para el análisis empírico de algoritmos de aprendizaje automático. La información sobre los conjuntos de datos elegidos se resume en la siguiente tabla 3.1 [27].

3.2. Modelos seleccionados

La idea de este Trabajo de Fin de Máster (TFM) era la de utilizar los conocimientos aprendidos durante el curso, por lo que se intentó escoger modelos vistos en clase, incluyendo algunos nuevos para su estudio y comparación. Los modelos elegidos fueron Random Forest (randomForest en caret), SVM lineal (svmLinear), SVM polinómico (svmPoly), regresión logística (glm), árbol de decisión (rpart), modelos de boosting (ada, xgbLinear), naive Bayes (naive Bayes) y k-nearest-neighbor (KNN). Todos menos ada y xgbLinear fueron vistos durante el curso.

3.2.1. Modelos de árboles de decisión

En este TFM hemos utilizado varios modelos basados en árboles de decisión (particionamiento recursivo): randomforest, rpart, Boosted Trees y xgboost. El particionamiento recursivo es una herramienta fundamental en la minería de datos. Nos ayuda a explorar la estructura de un conjunto de datos, mientras desarrolla reglas de decisión fáciles de visualizar para predecir un resultado categórico (árbol de clasificación) o continuo (árbol de regresión).

El paquete rpart proporciona un algoritmo del modelo ”árbol” tradicional. El paquete randomForest produce una gran cantidad de árboles por bootstrap (por eso se le denomina como un ’bosque’). Los paquetes ada y xgbLinear utilizan técnicas de boosting sobre árboles de decisión.

Los modelos de random forest gozaron de una gran popularidad al ser mucho más precisos en tareas de clasificación que los árboles de decisión tradicional. Esto se debe principalmente a la forma en que se realiza la partición en cada etapa, así como a los resultados de las muestras de bootstrap para agregar el resultado final.

Primero, en un bosque aleatorio, solo se selecciona un subconjunto de predictores y puede reducir el impacto de predictores fuertes. Esto se debe a que, una vez que un predictor fuerte hace una división en una etapa anterior, no hay mucho espacio para que otros predictores mejoren el ajuste. En este sentido, se puede identificar un patrón sistemático local.

También, en el caso que de que haya dos predictores altamente correlacionados, si se emplea un predictor para una división, es probable que el otro no se use en etapas posteriores. Sin embargo, si el primer predictor no está involucrado, este predictor puede desempeñar un papel, lo que puede resultar en una mejora adicional del ajuste.

Finalmente, uno de los principales beneficios de bootstrap es que se logra una varianza más baja.

Los dos primeros puntos pueden contribuir a un sesgo más bajo y el último a una varianza más baja para que su poder de predicción pueda mejorarse, aunque su interpretación puede no ser exhaustiva. Debido estas características, y a que los dos son modelos comúnmente usados para clasificación, se incluyó tanto el árbol de decisión como el random forest en este trabajo, para evaluar ambos con diferentes conjuntos de datasets y ver si existen en la práctica estas diferencias teóricas.

Los modelos de boosting ofrecen teóricamente ventajas similares a las obtenidas en random forest comparados con árboles de decisión tradicional.

Modelos de árboles de decisión

CART es un algoritmo de los modelos de árbol de decisión, el cual funciona dividiendo repetidamente los datos en múltiples subespacios, de modo que los resultados en cada subespacio final sean lo más homogéneos posible [21].

El modelo producido consiste en un conjunto de reglas utilizadas para predecir la variable de resultado, que puede ser: una variable continua para árboles de regresión o una variable categórica para árboles de clasificación. Las reglas de decisión generadas por el modelo predictivo CART generalmente se visualizan como un árbol binario [21].

El hiperparámetro que define el modelo es CP (parámetro de complejidad) y se usa para controlar el tamaño del árbol de decisión y para seleccionar el tamaño óptimo del árbol. Si el costo de agregar otra variable al árbol de decisión desde el nodo actual está por encima del valor de CP, entonces la construcción del árbol no continúa. También podríamos decir que la construcción de árboles no continúa a menos que disminuya la falta general de ajuste por un factor de cp. Para este trabajo hemos dado valores de entre (0,0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1) [21].

RandomForest

El modelo de RandomForest (rf) consiste en una gran cantidad de árboles de decisión individuales que operan como un conjunto [17]. Cada árbol individual en el bosque aleatorio realiza una predicción de clase y la clase con ”más votos” se convierte en la predicción del modelo. En este modelo la baja correlación entre modelos es clave. Los modelos no correlacionados pueden producir predicciones de conjunto que son más precisas que cualquiera de las predicciones individuales. La razón de este efecto es que los árboles se protegen entre sí de sus errores individuales (siempre que no se equivoquen constantemente en la misma dirección). Aunque algunos árboles pueden estar equivocados, muchos otros árboles estarán en lo correcto, por lo que, como grupo, los árboles se moverán en la dirección correcta.

El número de variables disponibles en cada nodo del árbol para realizar la partición es uno de los parámetros más importantes en random forest y en el paquete randomForest se conoce como ”mtry”. El valor por defecto que toma este parámetro depende de si trabajamos con un modelo de clasificación o regresión. Para los modelos de clasificación, el valor por defecto es

la raíz cuadrada del número de variables predictoras (redondeado hacia abajo). En cambio, para los modelos de regresión, es el número de variables predictoras dividido por 3 (redondeado hacia abajo). Debido a que el sujeto de estudio de este trabajo son los modelos de clasificación, utilizaremos una lista de de valores del parámetro mtry donde tomará valores comprendidos entre 2 y la longitud del dataset [17].

Los requisitos previos para que un random forest funcione bien son:

- Es necesario que haya alguna señal real en nuestras funciones para que los modelos creados con esas funciones funcionen mejor que las conjeturas aleatorias.
- Las predicciones (y, por lo tanto, los errores) hechas por los árboles individuales deben tener bajas correlaciones entre sí.

Boosted Classification Trees (ada)

Es un meta-algoritmo de aprendizaje automático. AdaBoost es adaptativo en el sentido de que los aprendices débiles posteriores se modifican en favor de aquellas instancias clasificadas erróneamente por clasificadores anteriores. AdaBoost es sensible a datos ruidosos y valores atípicos. En algunos problemas, puede ser menos susceptible al problema de sobreajuste que otros algoritmos de aprendizaje.

Cada algoritmo de aprendizaje tiende a adaptarse a algunos tipos de problemas mejor que otros, y generalmente tiene muchos parámetros y configuraciones diferentes para ajustar antes de lograr un rendimiento óptimo en un conjunto de datos. AdaBoost (con árboles de decisión como los alumnos débiles) a menudo aparece como el mejor clasificador listo para usar. Cuando se usa con el aprendizaje del árbol de decisión, la información reunida en cada etapa del algoritmo AdaBoost sobre la 'dureza'relativa de cada muestra de entrenamiento se alimenta al algoritmo de crecimiento del árbol de tal manera que los árboles posteriores tienden a centrarse en ejemplos más difíciles de clasificar.

Los hiperparámetros que definen a este modelo son: iter, maxdepth y nu. El parámetro iter indica el número de árboles de decisión. En general, el número de árboles agregados en este tipo de modelos suele ser alto para que el modelo funcione bien, aquí tomará valores comprendidos desde un mínimo de 20 y hasta un máximo de 500. El parámetro maxdepth indica la profundidad de los árboles, utilizaremos en este trabajo valores comprendidos de 1 a 5. Nu es el

hiperparámetro que ajusta los niveles de libertad, normalmente suelen usarse valores comprendidos entre 0 y 1.

eXtreme Gradient Boosting (xgbLinear)

XGBoost es una implementación específica del método Gradient Boosting que utiliza aproximaciones más precisas para encontrar el mejor modelo de árbol. Emplea una serie de ingeniosos trucos que lo hacen excepcionalmente exitoso, zparticularmente con datos estructurados. XGBoost [3] tiene ventajas adicionales: la capacitación es muy rápida y se puede paralelizar o distribuir en grupos. Para este modelo hemos utilizado los hiperparámetros: num_round, alpha, lambda y eta para calibrar el modelo. Num_round define el número de rondas de boosting, en nuestro caso hemos realizado dos rondas. Alpha es el término que regula L1 sobre pesos. Cuanto más aumentemos el valor de alpha más conservador será en modelo. En nuestro caso elegimos valores entre 0 y 1. Al igual que alpha, tenemos lambda que es el término de regulación para L2 sobre los pesos. Como en alpha, para lambda usamos valores entre 0 y 1. El hiper-parametro eta, reduce el tamaño del shrinkage utilizado en la actualización para evitar el sobreajuste. Después de cada shrinkage de aumento, podemos obtener directamente el peso de las nuevas caracteristicas, y eta reduce los pesos de las caracteristicas para hacer que el proceso de aumento sea más conservador. En nuestro trabajo eta oscilará entre los valores (0.01, 0.001, 0.0001) [3].

3.2.2. Support Vector Machines (svm)

Además de los árboles de decisión otro conjunto de modelos añadido fue SVM (Support Vector Machine -máquinas de vectores de soporte-). SVM son métodos de aprendizaje supervisados que se utilizan para problemas de clasificación, regresión o detección de outliers. Las ventajas de estos modelos son [24]:

- Efectividad en casos donde el número de dimensiones es mayor que el número de muestras.
- Utiliza un subconjunto de puntos de entrenamiento en la función de decisión (o también denominados vectores de soporte) por lo que son eficientes en memoria.
- Versatilidad: se pueden especificar diferentes funciones de Kernel (lineal, polinómica, etc)

para la función de decisión. Se proporcionan núcleos comunes, pero también es posible especificar núcleos personalizados.

Las desventajas de las máquinas de vectores de soporte incluyen:

- Si el número de características es mucho mayor que el número de muestras, evitar el ajuste excesivo al elegir las funciones del núcleo y el término de regularización es crucial.

- Los SVM no proporcionan directamente estimaciones de probabilidad, estas se calculan utilizando una costosa validación cruzada de (habitualmente) cinco veces.

Como podemos observar, las ventajas que los SVM aportan bastante interés como objeto de estudio en este trabajo, al ser modelos de demostrada efectividad para los tipos de conjuntos de datos escogidos.

La idea sobre la que se basan los modelos de SVM es la de encontrar un hiperplano (o conjunto de hiperplanos en un espacio de dimensionalidad muy alta (o incluso infinita) que mejor separe cada uno de las muestras en diferentes dominios o subconjuntos. Una buena separación entre las clases permitirá una clasificación correcta. El núcleo lineal (en caret, svmLinear), polinómico (svmPoly) y RBF o gaussiano son diferentes en cuanto al límite de decisión del hiperplano entre las clases. Las funciones del núcleo se utilizan para asignar el conjunto de datos original (lineal/no lineal) a un espacio dimensional superior con vistas a convertirlo en un conjunto de datos lineal. Por lo general, los núcleos lineales y polinómicos requieren menos tiempo.

Los hiperparámetros como C controlan cómo podría ser el movimiento del límite de decisión SVM. Cuanto mayor sea la C, se aplicará más penalización a SVM cuando clasificó de manera errónea y, por lo tanto, menor será el límite de decisión.

En este trabajo se han utilizado valores de C desde $10^{[-5,5]}$ para probar ambos modelos.

Para el modelo svmPoly, además hemos necesitado calibrar el modelo para los hiperparámetro degree y scale [24]. El parámetro degree controla la flexibilidad del límite de decisión. Los kernels de mayor grado producen un límite de decisión más flexible; en este trabajo hemos usado valores de 0 a 6. La principal ventaja del escalado (scale) [24] es evitar atributos en mayor número rangos que dominan aquellos en rangos numéricos más pequeños. Otra ventaja es evitar dificultades numéricas durante el cálculo, porque los valores del kernel generalmente dependen

de los productos internos de los vectores de características, por ello se han empleado valores comprendidos entre 10$\hat{}$[-15, .., 3].

3.2.3. Generalized Linear Model (glm)

Otro de los tipos de modelos utilizados han sido los modelos lineales generalizados o GLM. Los modelos lineales generalizados (GLM) extienden los modelos lineales de dos maneras [26]:

- Primero, los valores predichos están vinculados a una combinación lineal de las variables de entrada a través de una función de enlace inversa
- En segundo lugar, la función de pérdida al cuadrado se reemplaza por la desviación unitaria de una distribución en la familia exponencial (o más precisamente, un modelo de dispersión exponencial reproductiva (EDM)

El GLM generaliza la regresión lineal al permitir que el modelo lineal se relacione con la variable de respuesta a través de una función de enlace y al permitir que la magnitud de la varianza de cada medición sea una función de su valor predicho.

Los modelos lineales generalizados fueron formulados como una forma de unificar varios otros modelos estadísticos, incluida la regresión lineal, la regresión logística y la regresión de Poisson. Estos modelos se basan en un método de mínimos cuadrados iterativamente ponderado para la estimación de máxima probabilidad de los parámetros del modelo.

El problema de minimización se basa en donde está la penalización de regularización L2. Cuando se proporcionan pesos de muestra, el promedio se convierte en un promedio ponderado.

3.2.4. Naive Bayes (nb)

Los clasificadores naive Bayes son una familia de ”clasificadores probabilísticos” simples basados en la aplicación del teorema de Bayes con suposiciones de independencia fuertes entre las características, lo que alivia los problemas derivados de dimensionalidad. Los clasificadores Naive Bayes son altamente escalables y requieren una serie de parámetros lineales en el número de variables (características/predictores) en un problema de aprendizaje [19].

A pesar de las suposiciones a priori que son demasiado simplificadas, los clasificadores de Naive Bayes han funcionado bastante bien en muchas situaciones del mundo real, por ejemplo

en clasificar documentos y filtrar de spam.

Podemos ajustar los pocos hiperparámetros que tiene un modelo naive Bayes. El parámetro usekernel nos permite usar una estimación de densidad del núcleo para variables continuas versus una estimación de densidad guassiana, aquí hemos jugado con ambos valores (True y False). Adjust nos permite ajustar el ancho de banda de la densidad del núcleo, donde números más grandes significan una estimación de densidad más flexible. Aquí dimos a este parámetro valores de 0 a 5. El parámetro fL nos permite incorporar el Laplace más suave, tomamos un valor inicial de 0 y lo fuimos aumentando hasta 5.

3.2.5. k-Nearest Neighbors (k-nn)

El algoritmo k-nn es un algoritmo de aprendizaje automático supervisado simple y fácil de implementar que se puede utilizar para resolver problemas de clasificación y regresión. Este algoritmo supone que existen cosas similares en las proximidades. En otras palabras, cosas similares están cerca una de la otra. K-NN depende de que este supuesto sea lo suficientemente cierto para que el algoritmo sea útil, captura la idea de similitud (a veces llamada distancia, proximidad o cercanía) calculando la distancia entre puntos en un gráfico [8].

Calibramos el modelo mediante el hiperparámetro k. Este parámetro se refiere al número de vecinos más cercanos a incluir en la mayoría del proceso. En este trabajo toma valores comprendidos entre 1 y 20.

La principal desventaja de K-NN es la de ser significativamente más lento a medida que aumenta el volumen de datos la convierte en una opción poco práctica en entornos donde las predicciones deben hacerse rápidamente.

3.3. Estrategia de evaluación

La métrica principal en la que se ha basado la comparación entre modelos ha sido el F1 score, métrica más habitual para medir el rendimiento de un clasificador binario ya que balancea la precisión y sensibilidad (recall) mediante una media harmónica para ser una métrica más útil en caso de clases no balanceadas (algo común en clasificación binaria).

```
F1 = 2*precision*recall/(precision + recall)
```

La precisión se define como el número de verdaderos positivos (True Positives, TP) dividido entre el número total de predicciones positivas (TP + FP).

```
Precision = TP / (TP + FP)
```

La sensibilidad o recall se define como el número de verdaderos positivos (True Positives, TP) dividido entre el número total de positivos existentes en el dataset (TP + FN).

```
Recall = TP / (TP + FN)
```

Los conjuntos de datos utilizados tenían variables a predecir de tipo binario, pero su traducción a ”positivo” y ”negativo” no era siempre automática. Como el objetivo de este trabajo era que fuese lo más automatizado posible, a la hora de evaluar cada uno de los modelos entrenados con cada uno de los conjuntos de datos se tuvo que adjudicar qué valor se consideraba como positivo y cuál como negativo, ya que como se ve en las definiciones anteriores esto afecta a la métrica de comparación (F1 score).

Pongamos un ejemplo muy simple para explicar los pasos seguidos -que son los mismos que se realizan para cada uno de los datasets con cada uno de los modelos-:

- Un dataset tiene como valores categóricos clasificar entre ”a” y ”b”.
- Los datos de train o entrenamiento se entrenan por un modelo X .
- El subconjunto de test o evaluación usa el modelo X para predecir la variable ”label”. Esa variable label, que queremos predecir, tiene como valores a y b.
- Sin embargo el modelo X al realizar la predicción devolverá un valor entre 0 y 1, que dependiendo de dónde se ponga el threshold, se traducirá como una clase u otra. Es decir, si por ejemplo indicamos que nuestro valor positivo es a (1) y nuestro valor negativos es b (0) y el modelo predice para una muestra x1 una probabilidad de 0.3, x2 de 0.2 y x3 de 0.7. Si establecemos que nuestros threshold es 0.5, el modelo dirá que las muestras x1 y x2 predichas tienen pinta de ser negativas, es decir que pertenezcan a la clase b, y la muestra predicha x3 pertenezca a la clase a o sean positivas.
- Dependiendo de cómo se asignen positivo/negativo a cada uno de los valores de la clase, el F1 score del modelo X será. Es decir, si a la clase a se atribuía al positivo y a la clase b

el negativo, se obtienen unos valores de recall, precisión y F1 diferentes a si a la clase a se le define como negativo y a la clase b como positivo.

Para resolver este problema, lo que se hizo fue calcular el F1 score con ambas asignaciones para todos los modelos y todos los conjuntos de datasets y se escogió aquella asignación que daba un F1 score más alto. De esta manera se aseguraba que la asignación hecha era la más óptima.

El código que recoge esta estrategia seguida es el siguiente:

```
pred <- prediction(probs,actuals,
           label.ordering = c(negative\_class,positive\_class))
perf <- performance(pred,''f")
bestF1ScoreIndex <- which.max(perf@"y.values"[[1]])
f1 <- perf@''y.values''[[1]][bestF1ScoreIndex]
```

Capítulo 4

Resultados

Para cada uno de los 20 datasets, y para cada uno de los 9 modelos, se realizó un entrenamiento, un ajuste de hiperparámetros, y una evaluación, obteniendo como resultado pares modelo-dataset con su correspondiente F1 score.

Para que los resultados obtenidos fueran generalizables a datasets más allá de los usados, se han clasificado por sus características:

- Por número de variables (menos de 10 variables y más de 10 variables)
- Por número de ejemplos (menos de 500 ejemplos y más de 500 ejemplos)
- Por tipo de variables (todas categóricas, todas numéricas y mezcla de categóricas y numéricas)

El número de datasets de cada tipo se pueden ver en la gráfica 4.1.

4.1. Resultados para datasets de menos de 10 variables

En total había 11 datasets con menos de 10 variables. De estos datasets había 8 con menos de 500 ejemplos (3 con más de 500 ejemplos), 2 con todas las variables categóricas y 5 con todas numéricas.

Los resultados obtenidos por modelo se ven en la gráfica 4.2.

Se puede observar que para este tipo de grupos de datasets los modelos que mejor funcionan son svmPoly y rf. Son resultados que no sorprenden dado que son modelos que funcionan bien

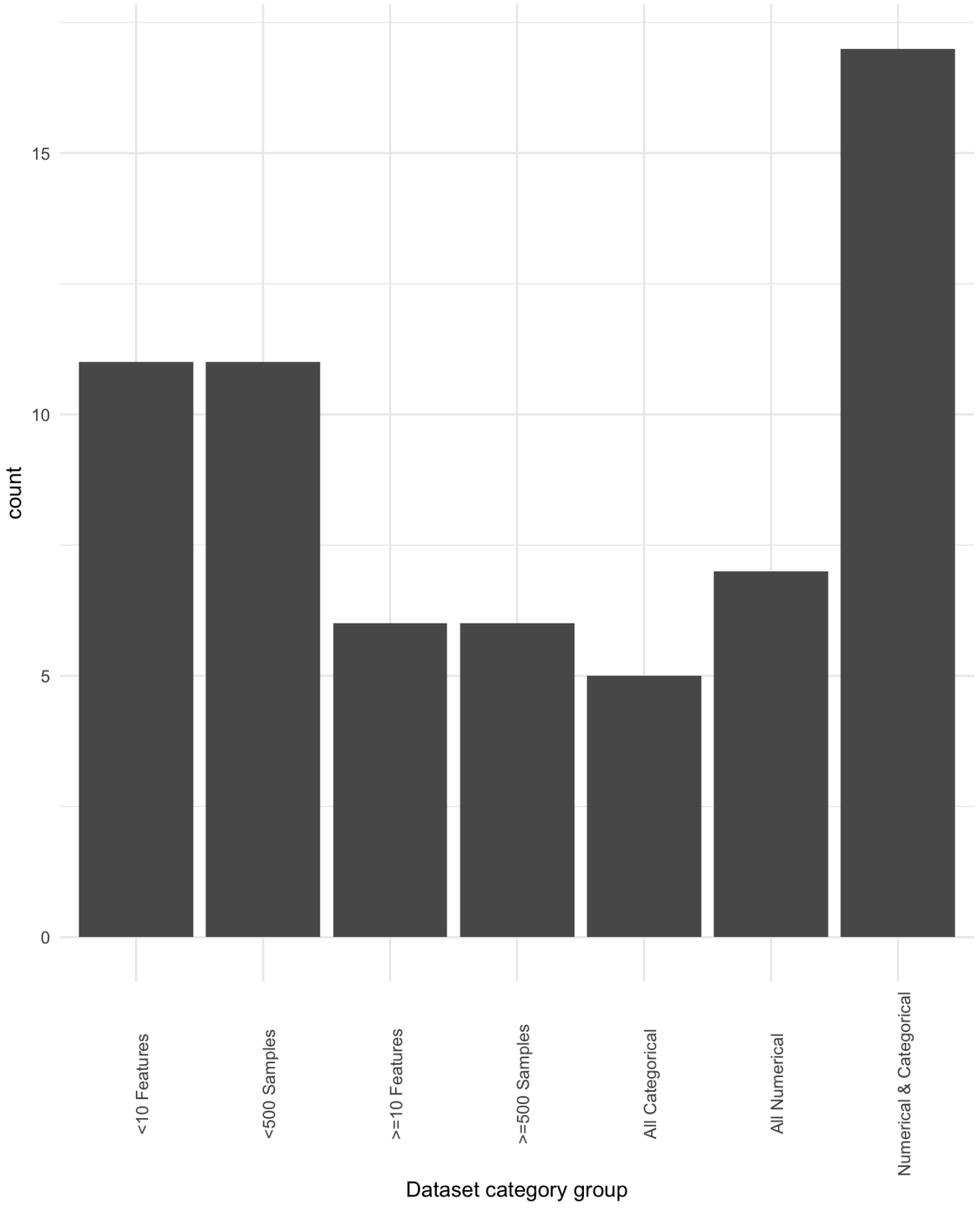

Figura 4.1: Número de datasets según categoría

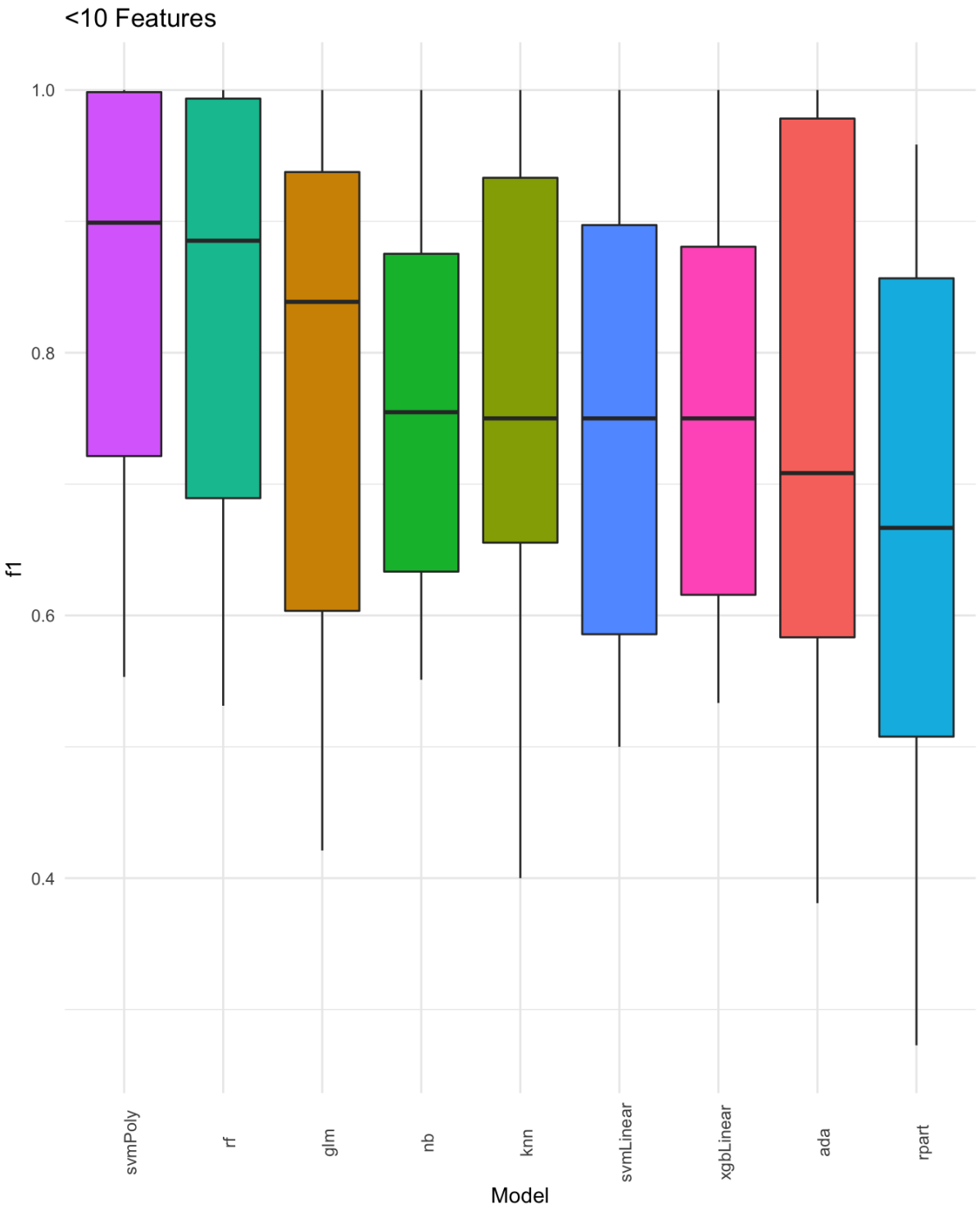

Figura 4.2: Datasets con menos de 10 variables

tanto con pocas como con muchas variables. Sin embargo cabe destacar que la diferencia de rendimiento entre estos dos modelos y glm no es tan grande, por lo que dependiendo del caso de uso se podría elegir glm, ya que es un modelo más simple.

A destacar sería los bajos valores de F1 que se obtienen con rpart. Este resultado sorprende puesto que rpart debería funcionar, si acaso, para datasets no muy grandes, y en especial para este tipo de conjuntos de datos de menos de 10 variables.

Parece ser que los datasets no eran suficientemente complejos para que xgbLinear o ada produjeran un resultado mejor frente a modelos más simples (probablemente estén ya sobre-ajustando).

4.2. Resultados para datasets de más de 10 variables

En total había 6 datasets con más de 10 variables. De estos datasets había 3 con menos de 500 ejemplos (3 con más de 500 ejemplos), 3 con todas las variables categóricas y 2 con todas numéricas.

Los resultados obtenidos por modelo se ven en la gráfica 4.3.

Se puede observar que en general todos los modelos dan resultados más similares, en torno a 0.7 de F1 score. A excepción de rpart y, especialmente, ada, que está por debajo de 0.6. En el caso de rpart, es esperado porque para datasets con muchas variables no es un modelo lo suficientemente complejo.

4.3. Resultados para datasets de menos de 500 ejemplos

En total había 11 datasets con menos de 500 ejemplos. De estos datasets había 8 con menos de 10 variables (3 con más de 10 variables), 1 con todas las variables categóricas y 6 con todas numéricas.

Los resultados obtenidos por modelo se ven en la siguiente gráfica 4.4.

Se puede observar que: para este tipo de conjuntos de datos svmPoly sigue siendo el modelo más óptimo pero aquí se da que nb resulta ser mejor que rf, al contrario que en los anteriores tipos de datos.

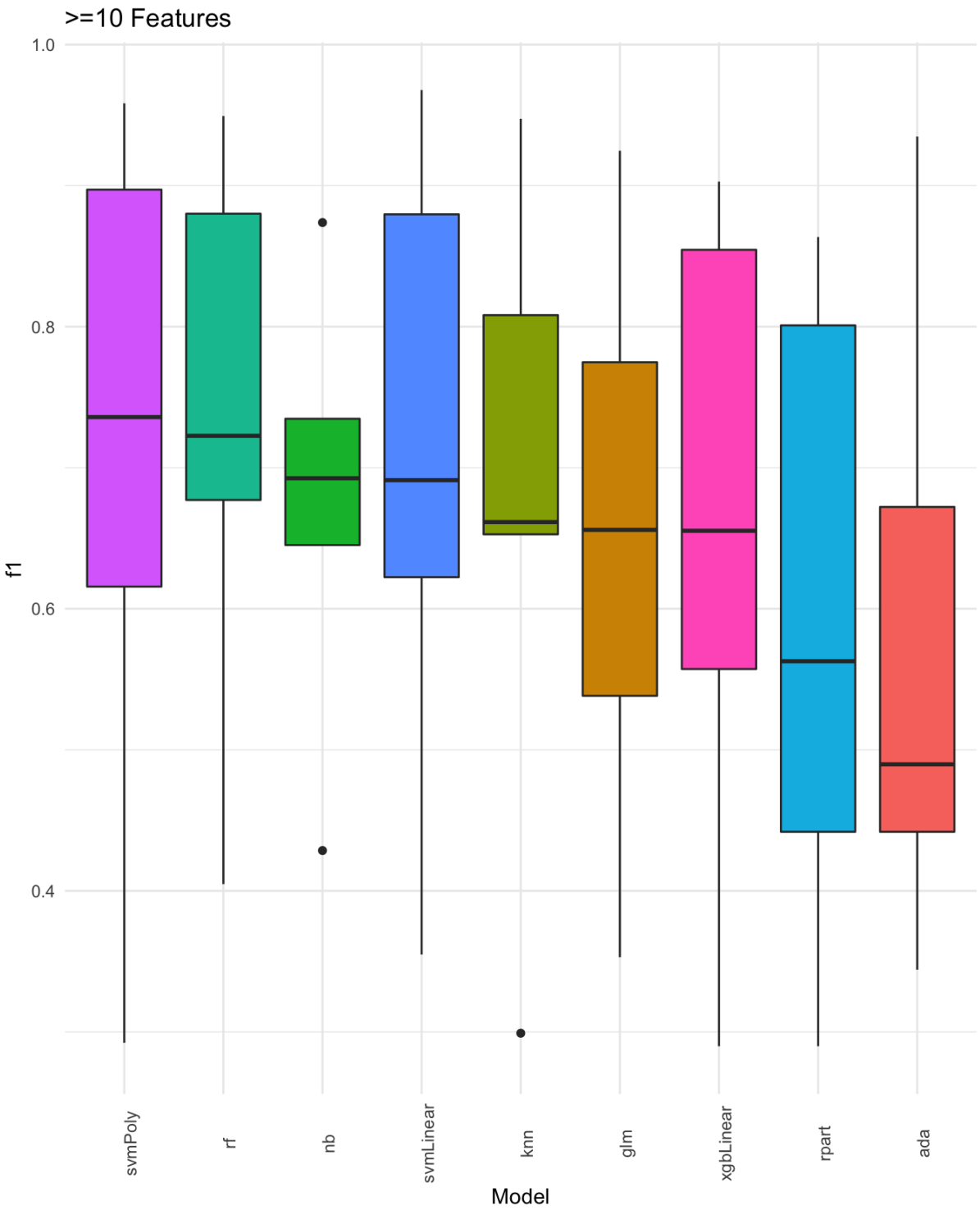

Figura 4.3: Datasets con más de 10 variables

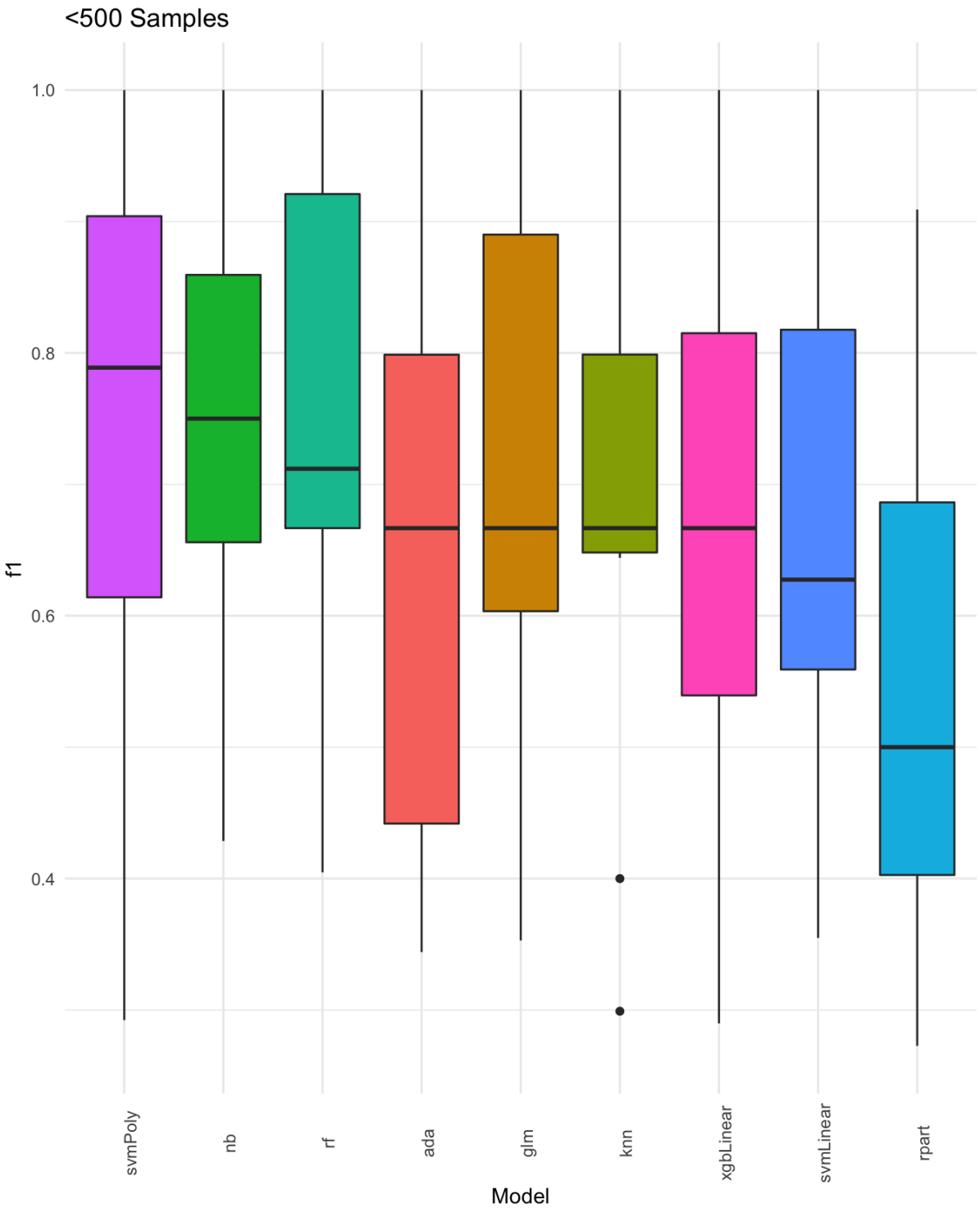

Figura 4.4: Datasets con menos de 500 variables

Este resultado sorprende, ya que nb es un modelo que habitualmente se recomienda para ejercicios de clasificación de texto, los cuáles contienen multitud de variables.

4.4. Resultados para datasets de más de 500 ejemplos

En total había 6 datasets con más de 500 ejemplos. De estos datasets había 3 con menos de 10 variables (3 con más de 10 variables), 4 con todas las variables categóricas y 1 con todas numéricas.

Los resultados obtenidos por modelo se ven en la siguiente gráfica 4.5.

Se puede observar que: svmPoly y rf en este caso son los dos mejores modelos a usar, seguidos de k-nn. Cabe destacar, de nuevo, que el modelo de nb no es el que mejor funciona con conjuntos de datos con muchas muestras.

El modelo xgbLinear ha mejorado su posición con respecto a ránkings anteriores, lo cual tiene sentido ya que es un modelo que tiende a funcionar mejor con muchas variables y muchas muestras.

4.5. Resultados para datasets con todas las variables categóricas

En total había 5 datasets con todas las variables categóricas. De estos datasets había 2 con menos de 10 variables (3 con más de 10 variables), 1 con menos de 500 ejemplos y 4 con más de 500 ejemplos.

Los resultados obtenidos por modelo se ven en la siguiente gráfica 4.6.

Se puede observar que: cuando se trabaja con conjuntos de datos donde todas las variables son de tipo categórico, el modelo de ada no suele funcionar bien (o requiere de un ajuste mayor). En este caso vemos que a diferencia de ránkings anteriores los modelos glm y svmLinear aparecen entre los que generan mejores resultados.

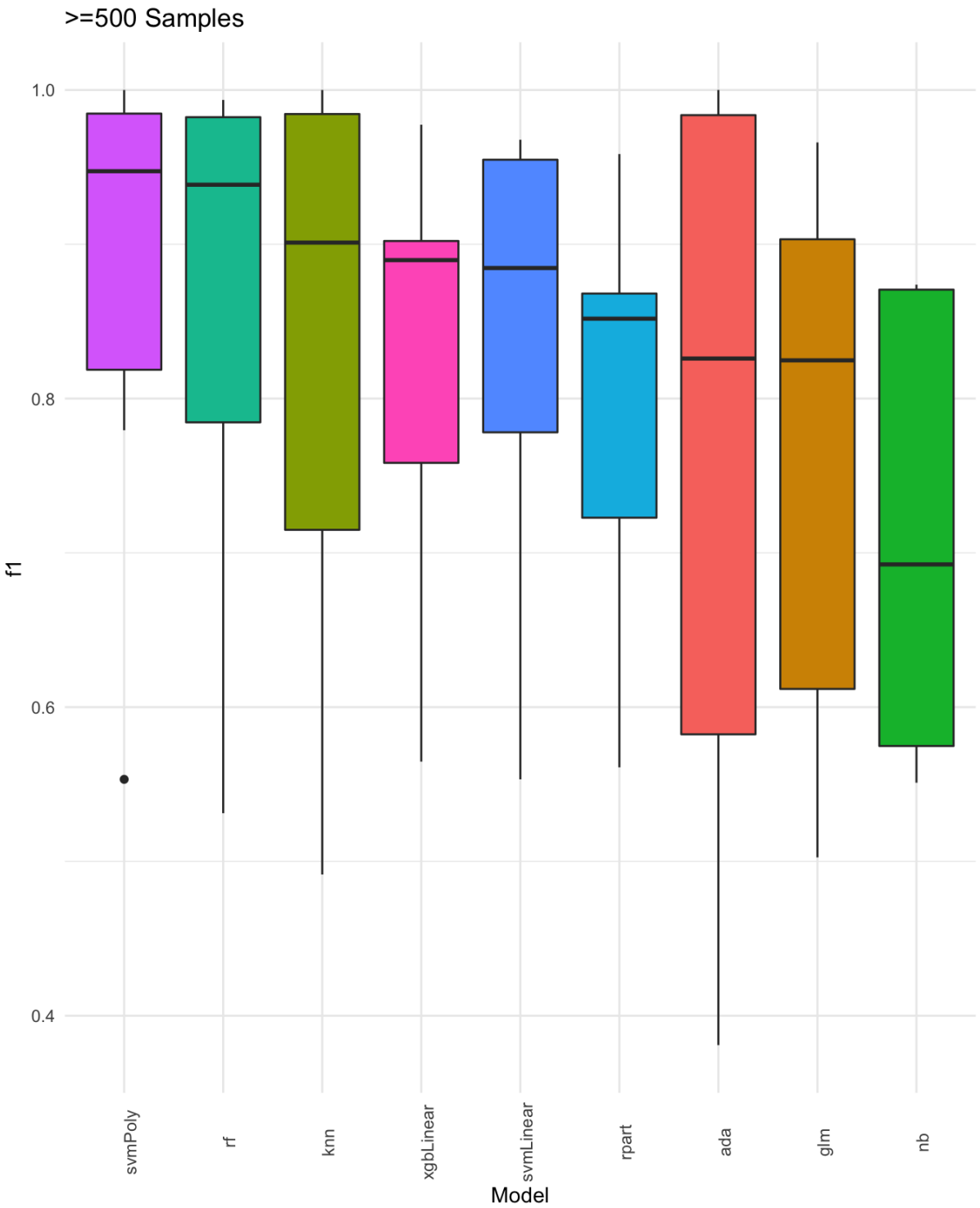

Figura 4.5: Datasets con más de 500 variables

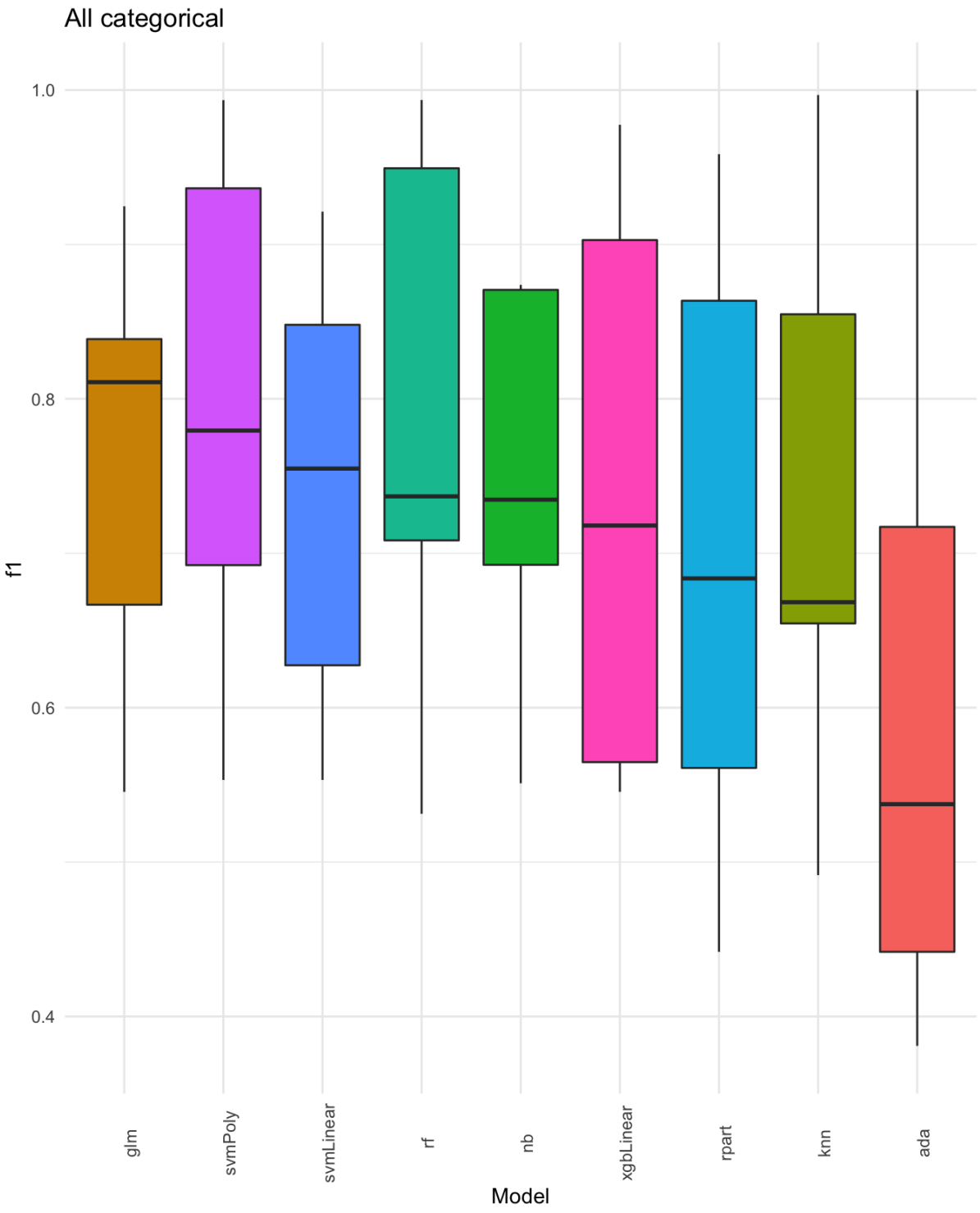

Figura 4.6: Datasets con todas las variables categóricas

4.6. Resultados para datasets con todas las variables numéricas

En total había 7 datasets con todas las variables numéricas. De estos datasets había 5 con menos de 10 variables (2 con más de 10 variables), 6 con menos de 500 ejemplos y 1 con más de 500 ejemplos.

Los resultados obtenidos por modelo se ven en la siguiente gráfica 4.7.

Se puede observar que: en general los resultados son bajos si los comparamos con las anteriores categorías, en especial el modelo de rpart, donde se puede ver que el valor de F1 está en torno a 0.5. El modelo de nb parace que es el que mejor se adapta a estos tipos de datasets, pero la diferencia no es decisiva si la comparamos con el resto de modelos (especialmente si tenemos en cuenta la dispersión).

4.7. Resultados para datasets con una mezcla de variables categóricas y numéricas

En total había 17 datasets con mezcla de variables categóricas y numéricas. De estos datasets había 11 con menos de 10 variables (6 con más de 10 variables), 11 con menos de 500 ejemplos y 6 con más de 500 ejemplos.

Los resultados obtenidos por modelo se ven en la siguiente gráfica 4.8.

Se puede observar que: todos los modelos escogidos tienen un valor de F1 bastante alto en general. Como en categorias anteriores svmPoly y rf son los modelos que mejor funcionan.

4.8. Resultados por modelo

En los resultados por modelo se puede observar un mismo patrón de resultados que se aplica en la mayor parte de los modelos. En su gran mayoría los modelos obtienen los mejores resultados con datasets de más de 500 muestras. Esto coincide con lo que cabe esperar teóricamente, ya esto permite a los modelos desarrollar su complejidad.

A primera vista sorprende los resultados para los datasets cuando solo tienen variables numéricas, que se sitúan en los datasets con peor rendimiento (aspecto que no concuerda con

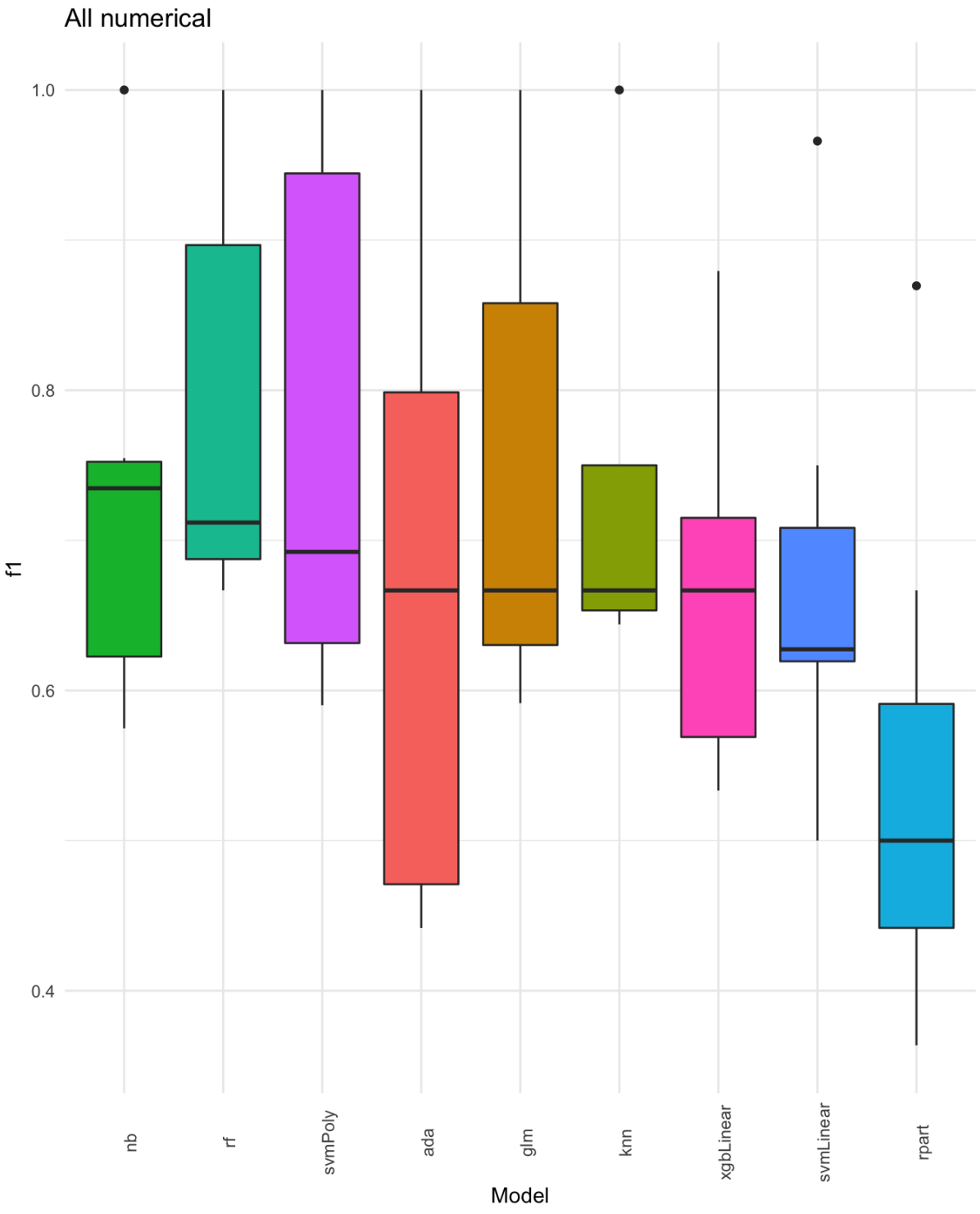

Figura 4.7: Datasets con todas las variables numéricas

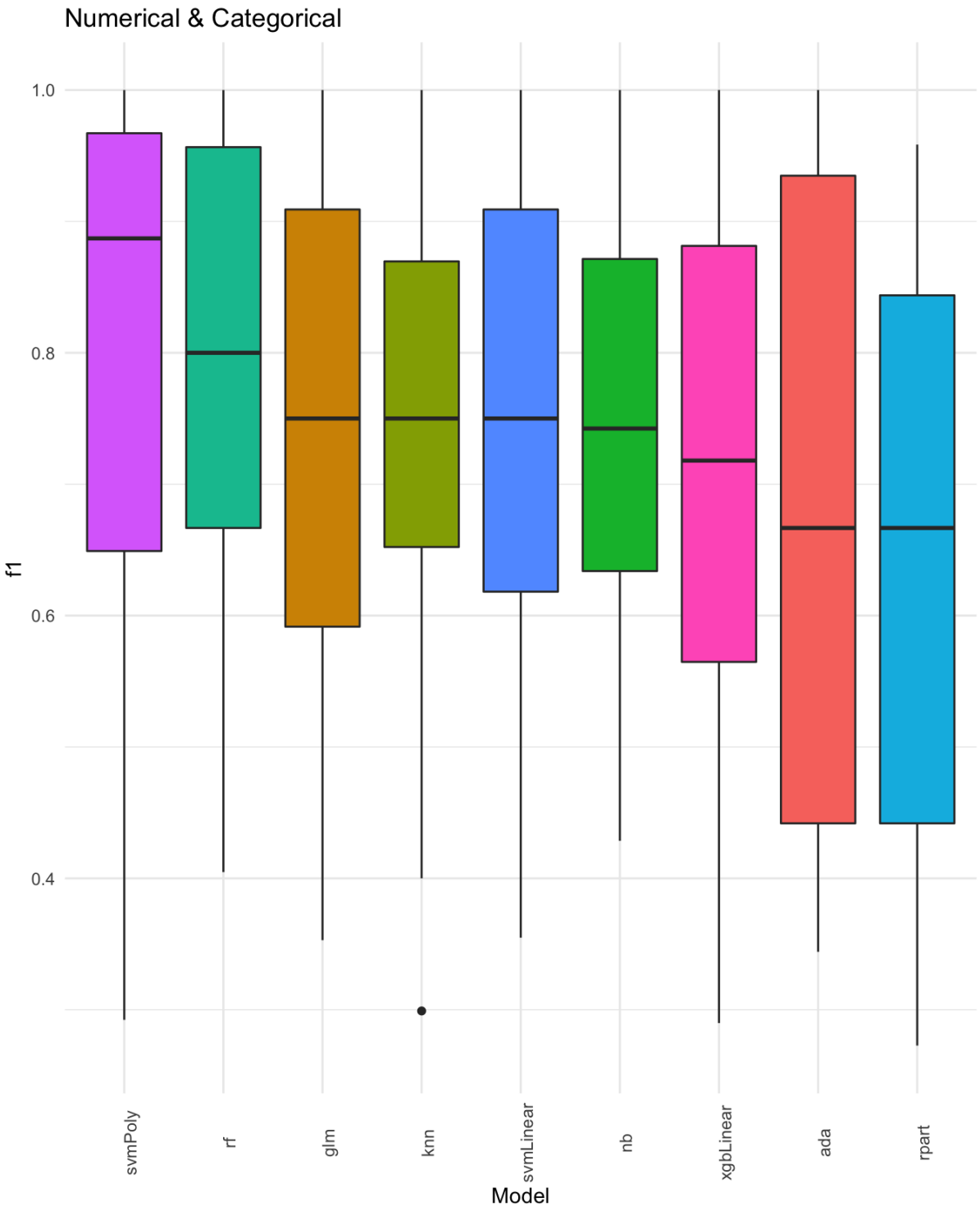

Figura 4.8: Datasets con variables categóricas y numéricas

lo esperado teóricamente). Pero esto se explica debido a que el subconunto de datasets que sólo contiene variables numéricas tienen en su mayoría menos de 10 variables y menos de 500 muestras.

Otro patrón a destacar es que los modelos en general tienen un mayor rendimiento en datasets con menos de 10 features. La explicación a esto es que se tratan de problemas más simples y un menor número de features está más en concordancia con el número de muestras que tienen de media los datasets (en mediana todos los conjuntos de datos tienen menos de 500 muestras).

Existe un modelo con resultados que no concuerdan con los patrones citados anteriormente. Se trata del modelo nb. El modelo de naive bayes obtiene un rendimiento similar con todos los diferentes tipos de datasets, lo cual pude interpretarse como una ventaja a la hora de elegir este modelo como primera opción ante un dataset de características desconocidas.

A continuación las siguientes gráficas sobre los resultados obtenidos.

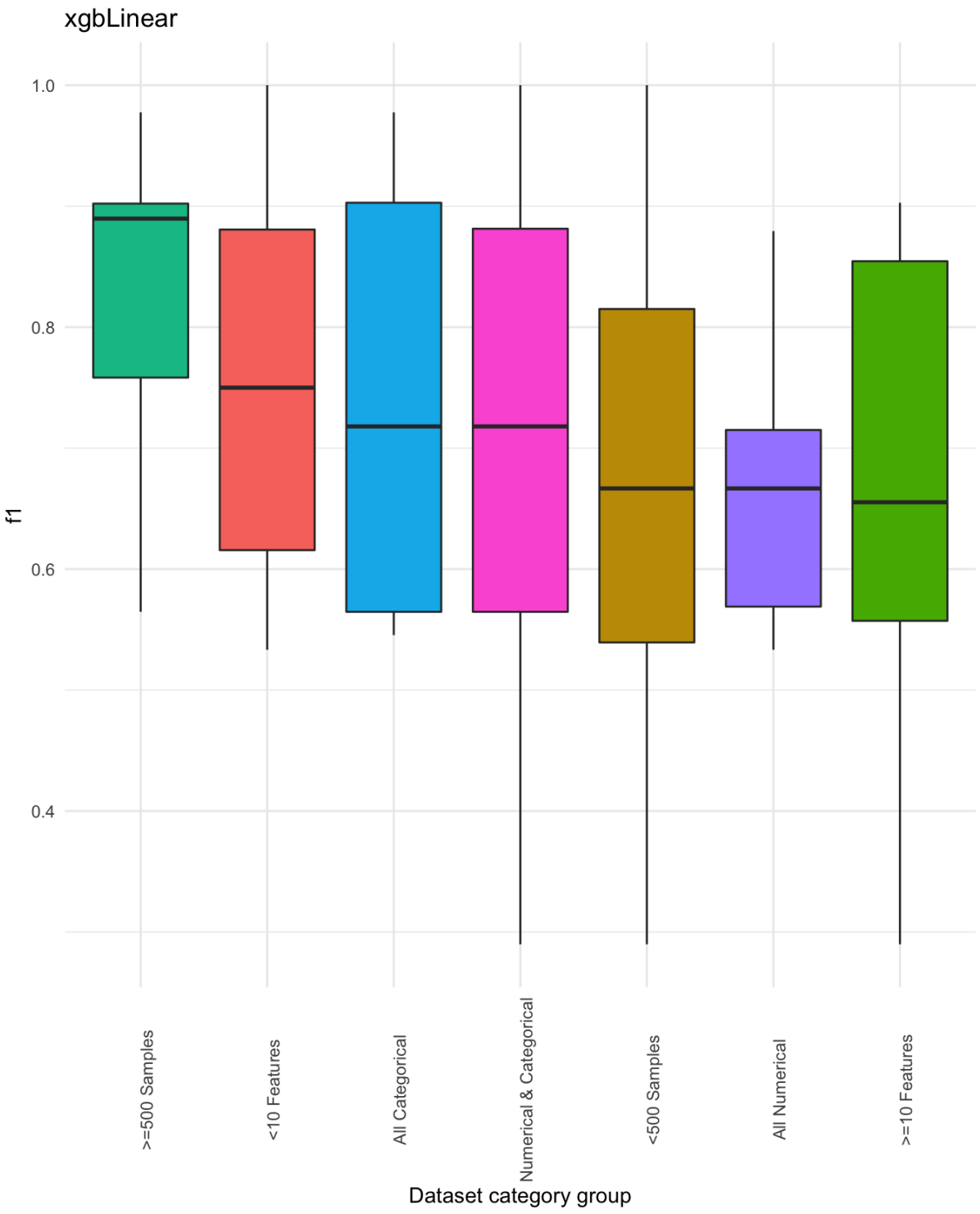

Figura 4.9: Resultados de xgbLinear con todos los distintos tipos de datasets

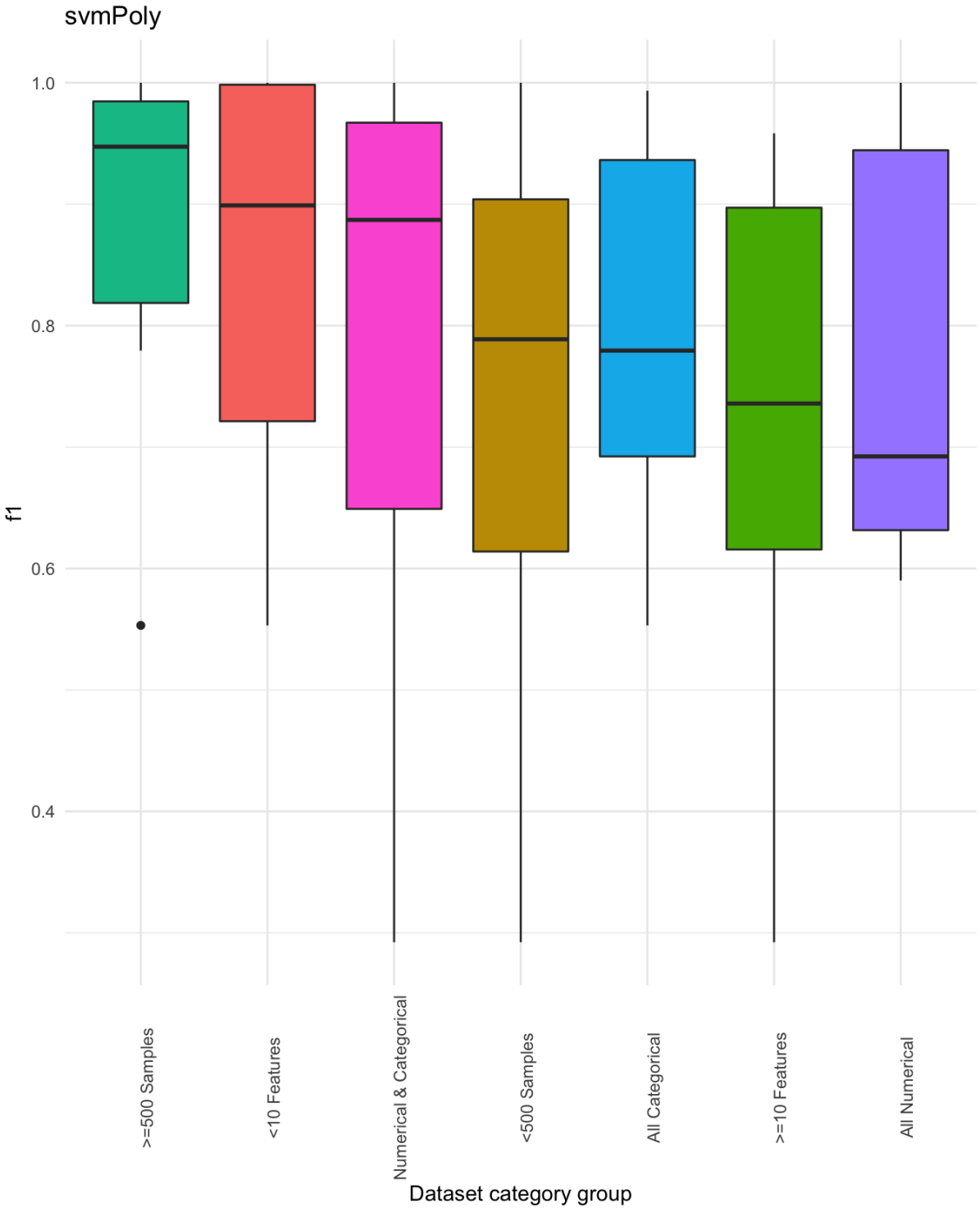

Figura 4.10: Resultados de SVMPoly con todos los distintos tipos de datasets

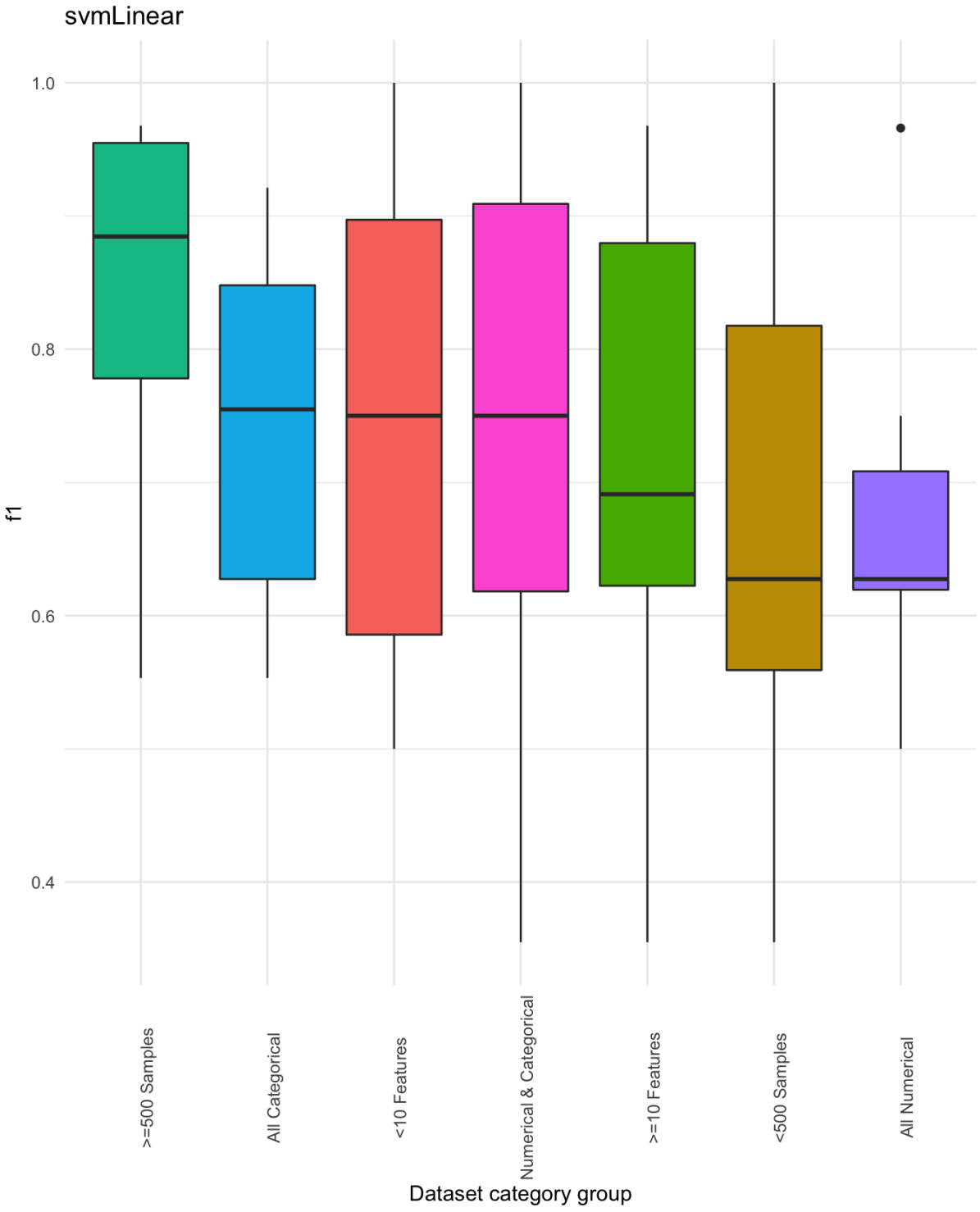

Figura 4.11: Resultados de SVMLinear con todos los distintos tipos de datasets

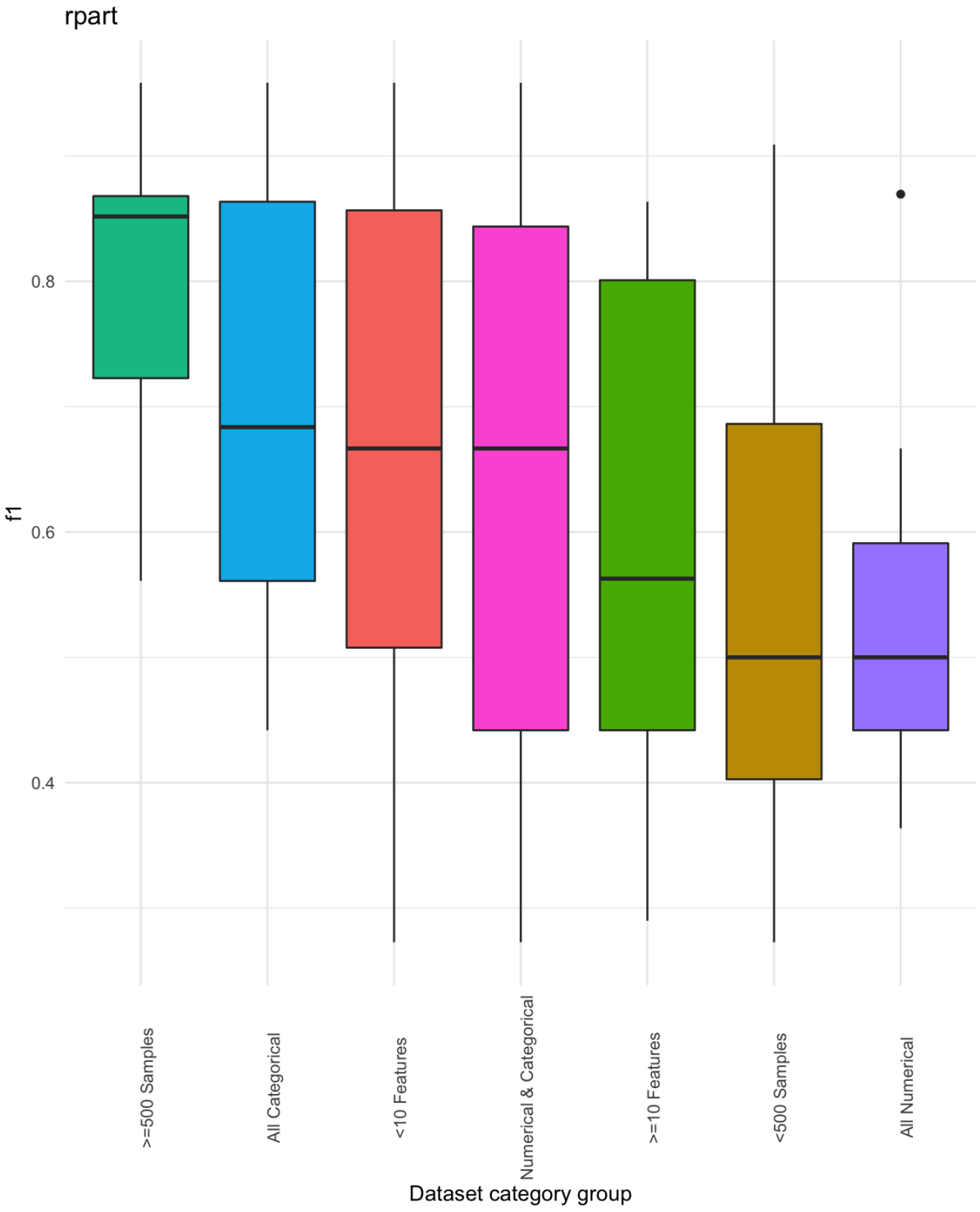

Figura 4.12: Resultados de rpart con todos los distintos tipos de datasets

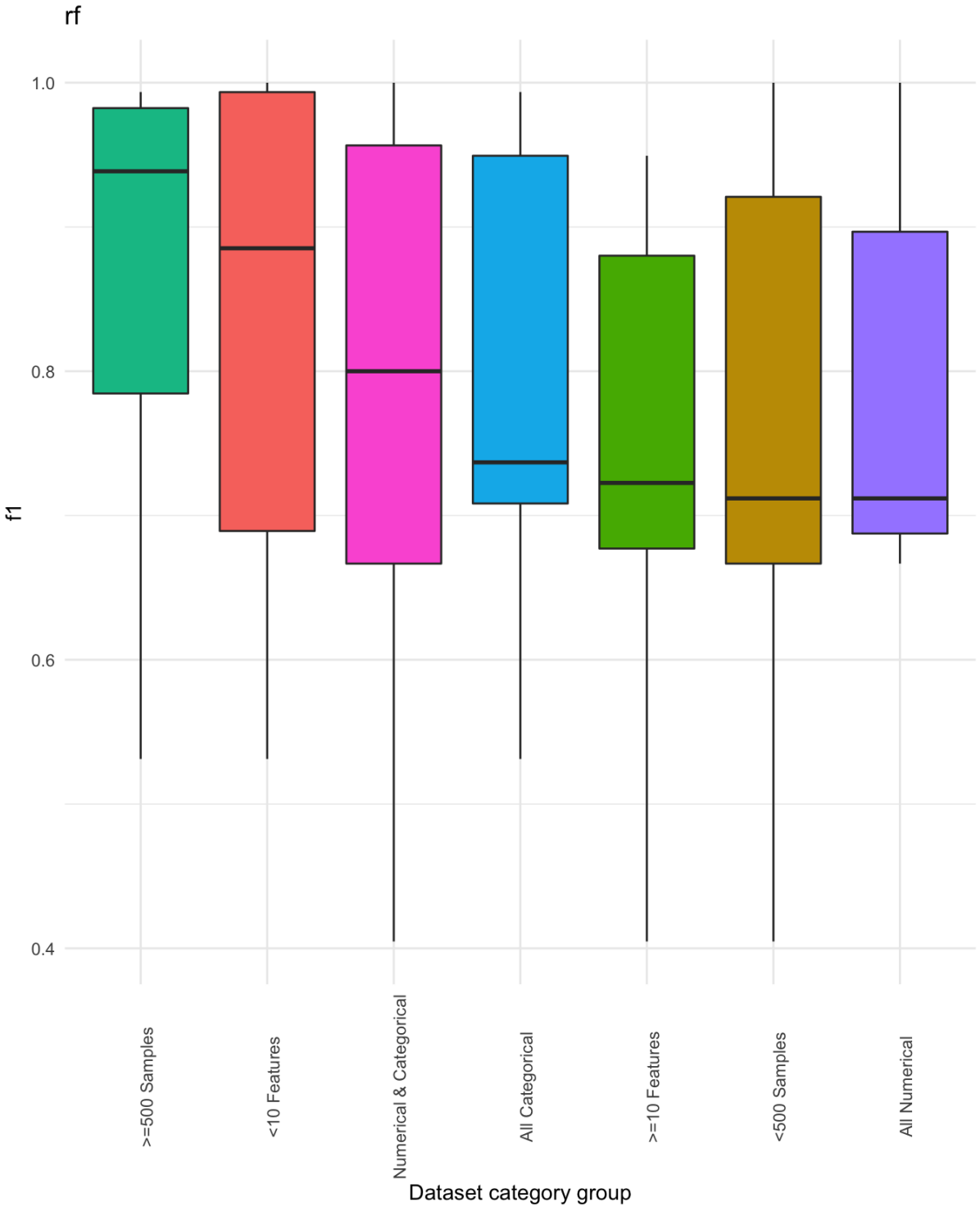

Figura 4.13: Resultados de randomForest con todos los distintos tipos de datasets

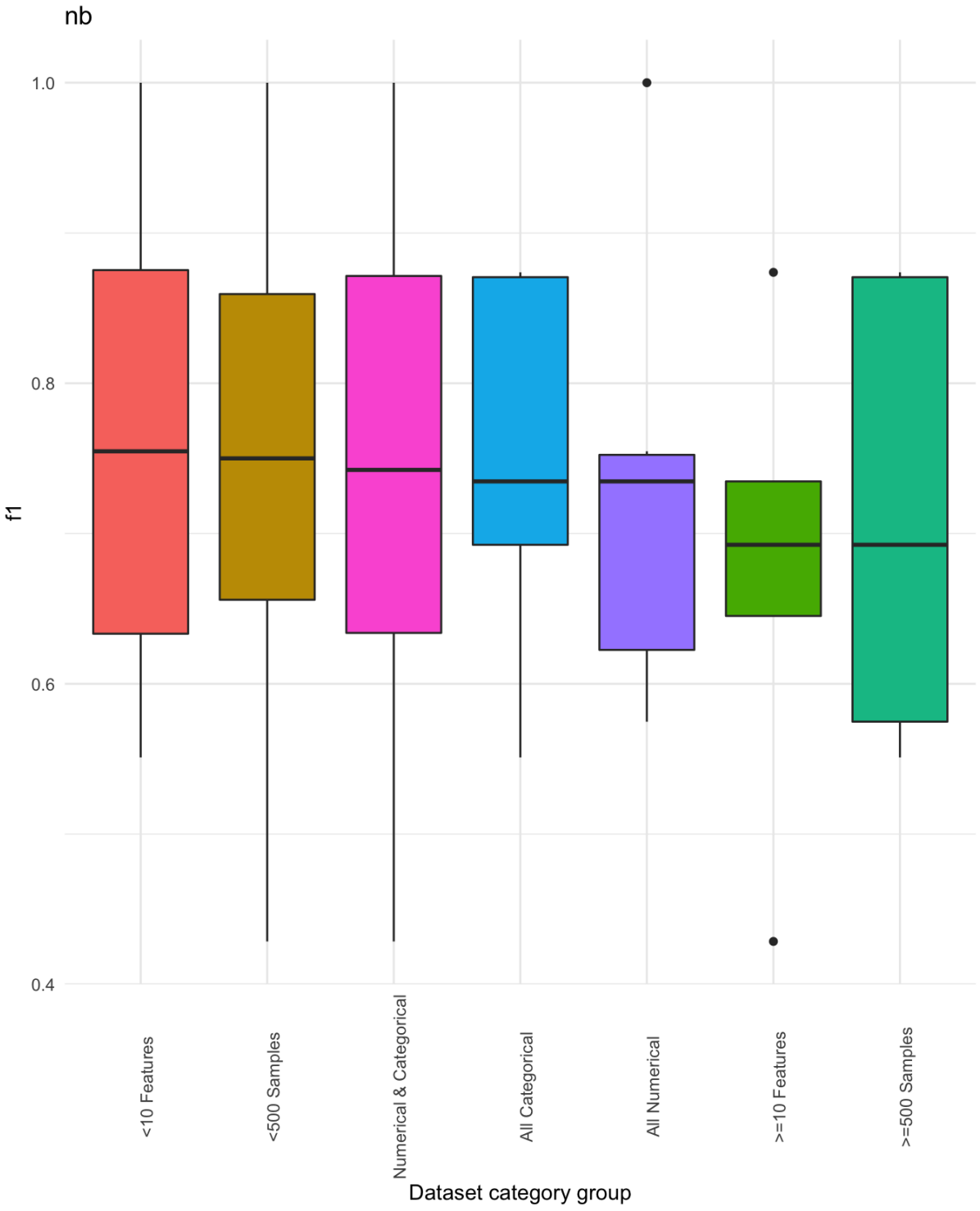

Figura 4.14: Resultados de naive bayes con todos los distintos tipos de datasets

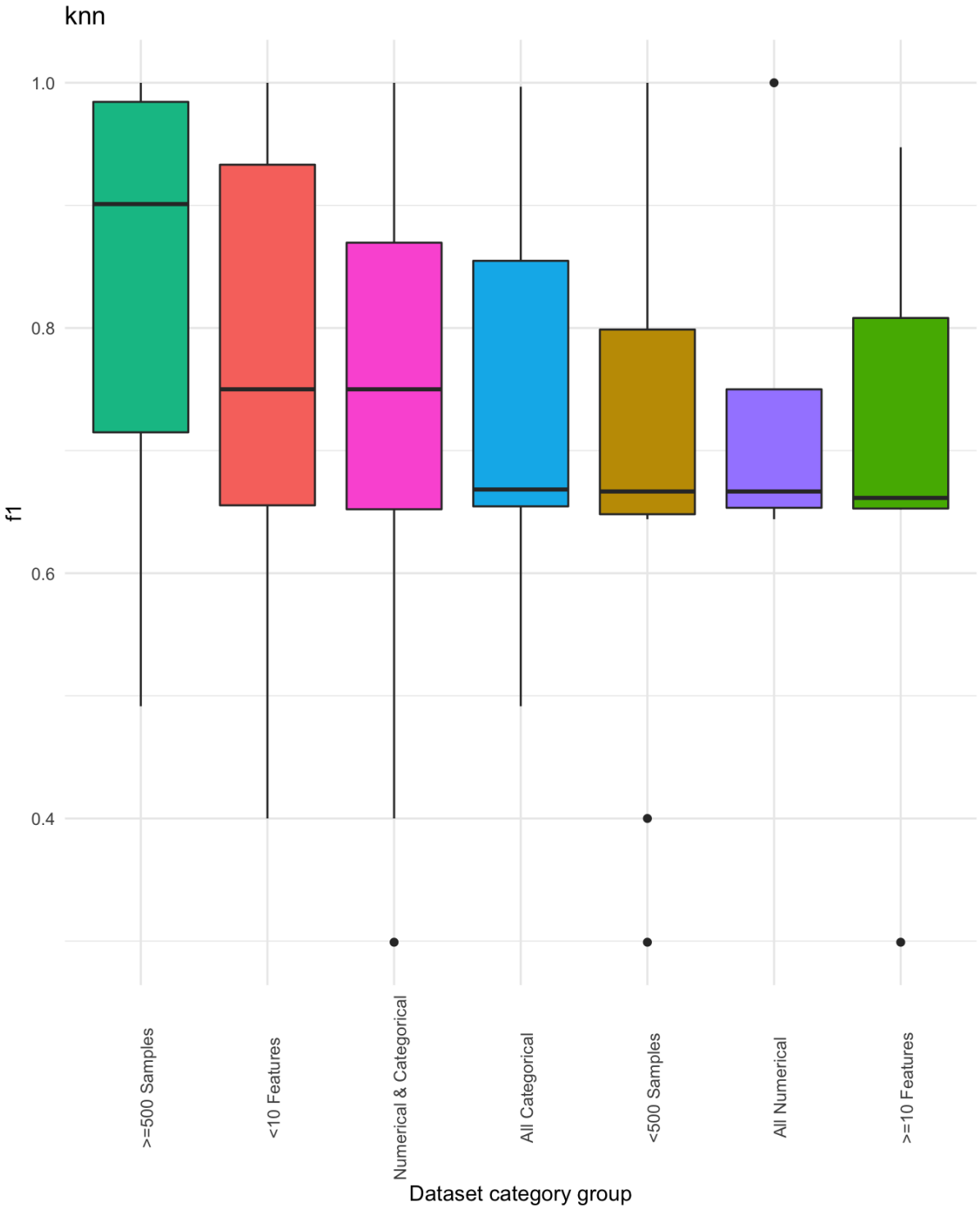

Figura 4.15: Resultados de KNN con todos los distintos tipos de datasets

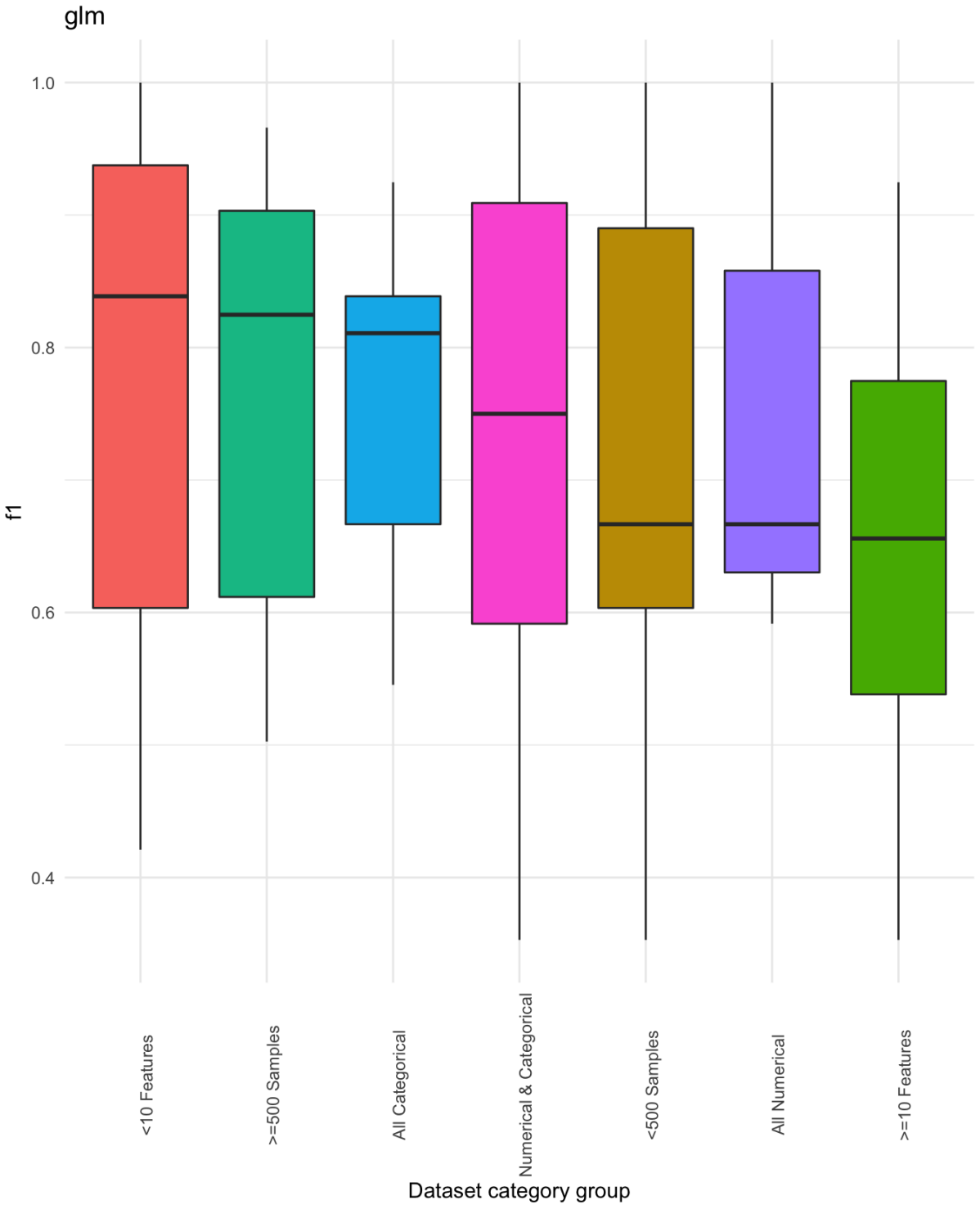

Figura 4.16: Resultados de GLM con todos los distintos tipos de datasets

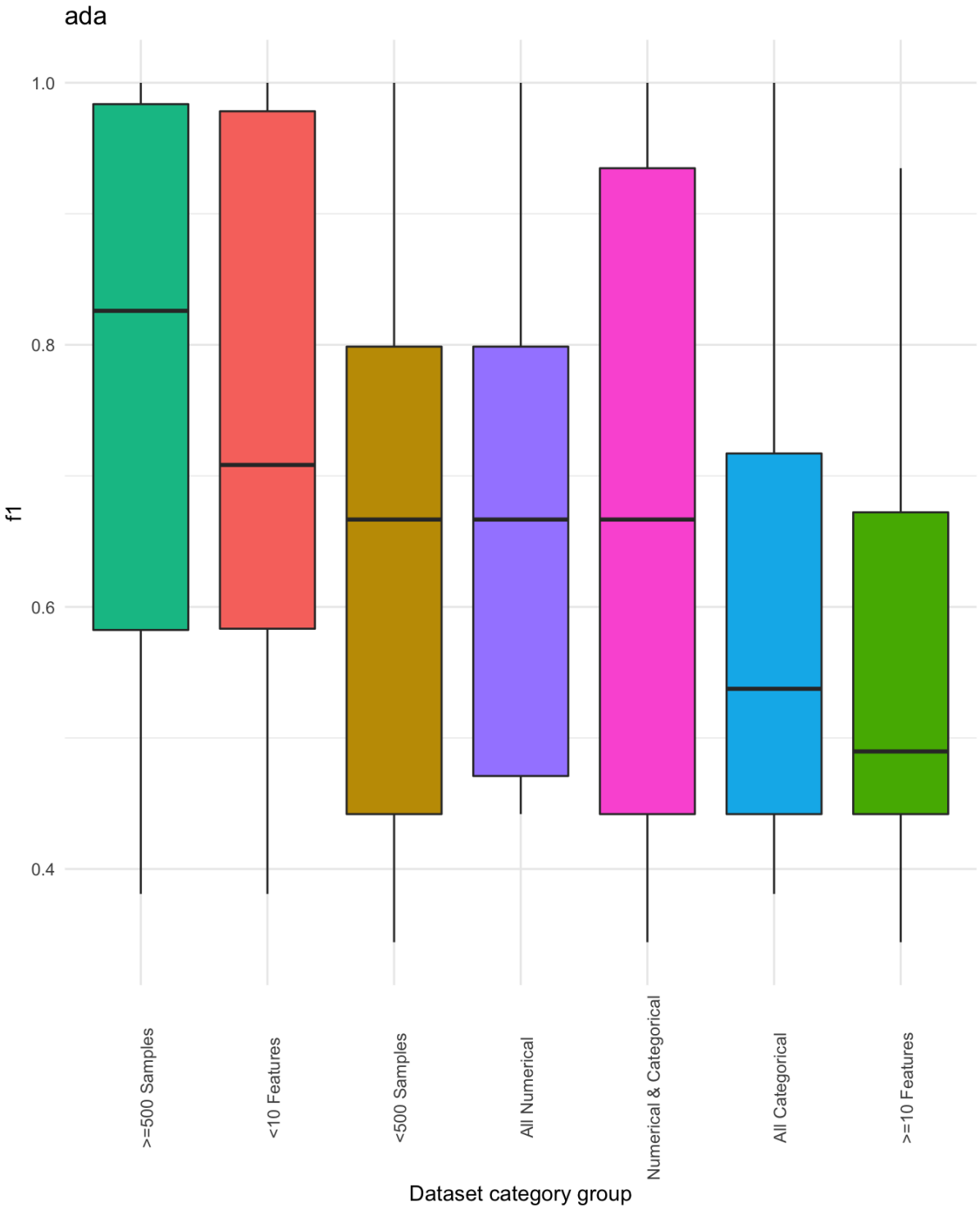

Figura 4.17: Resultados de ADA con todos los distintos tipos de datasets

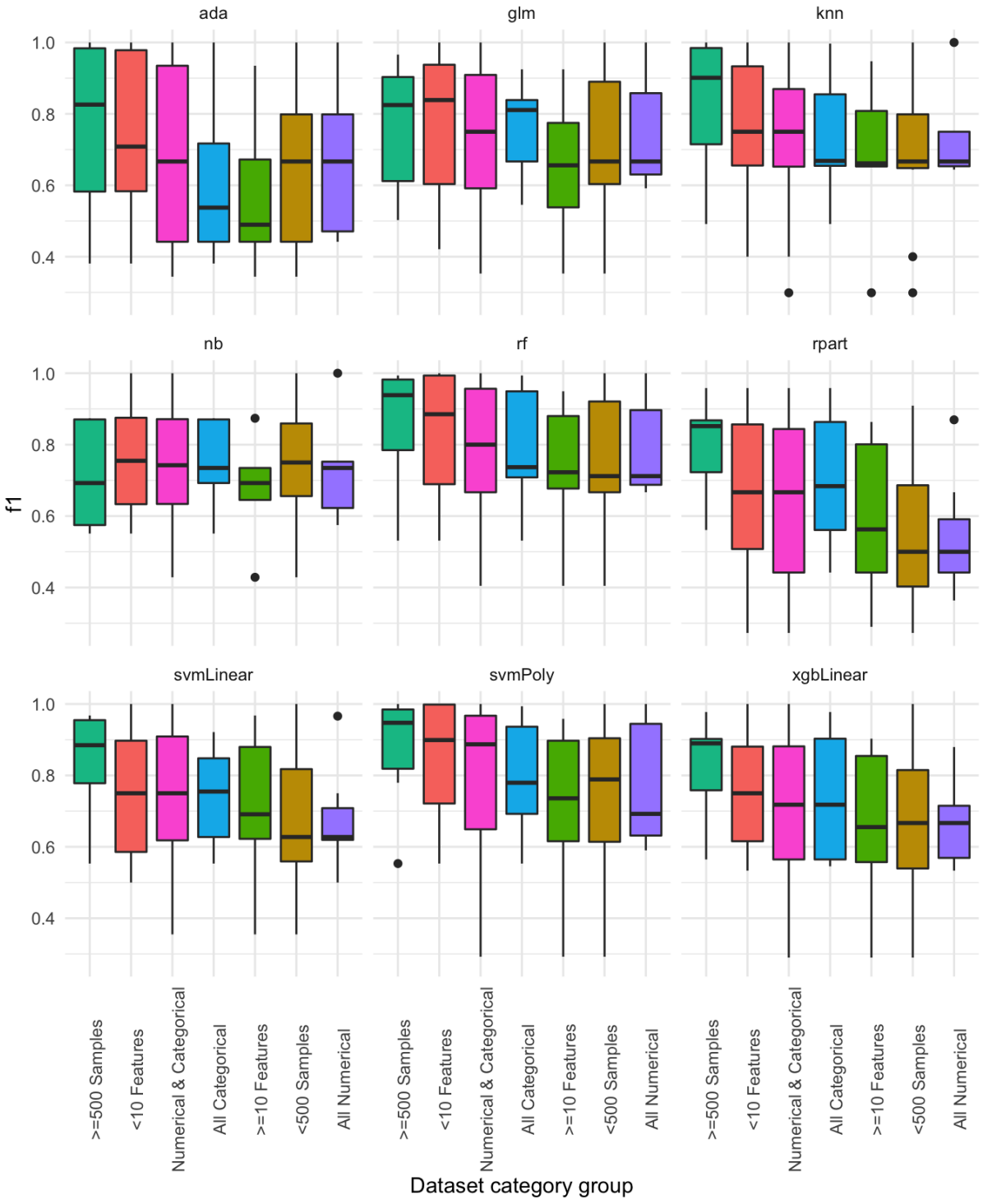

Figura 4.18: Compación de todos los modelos según los distintos tipos de datasets

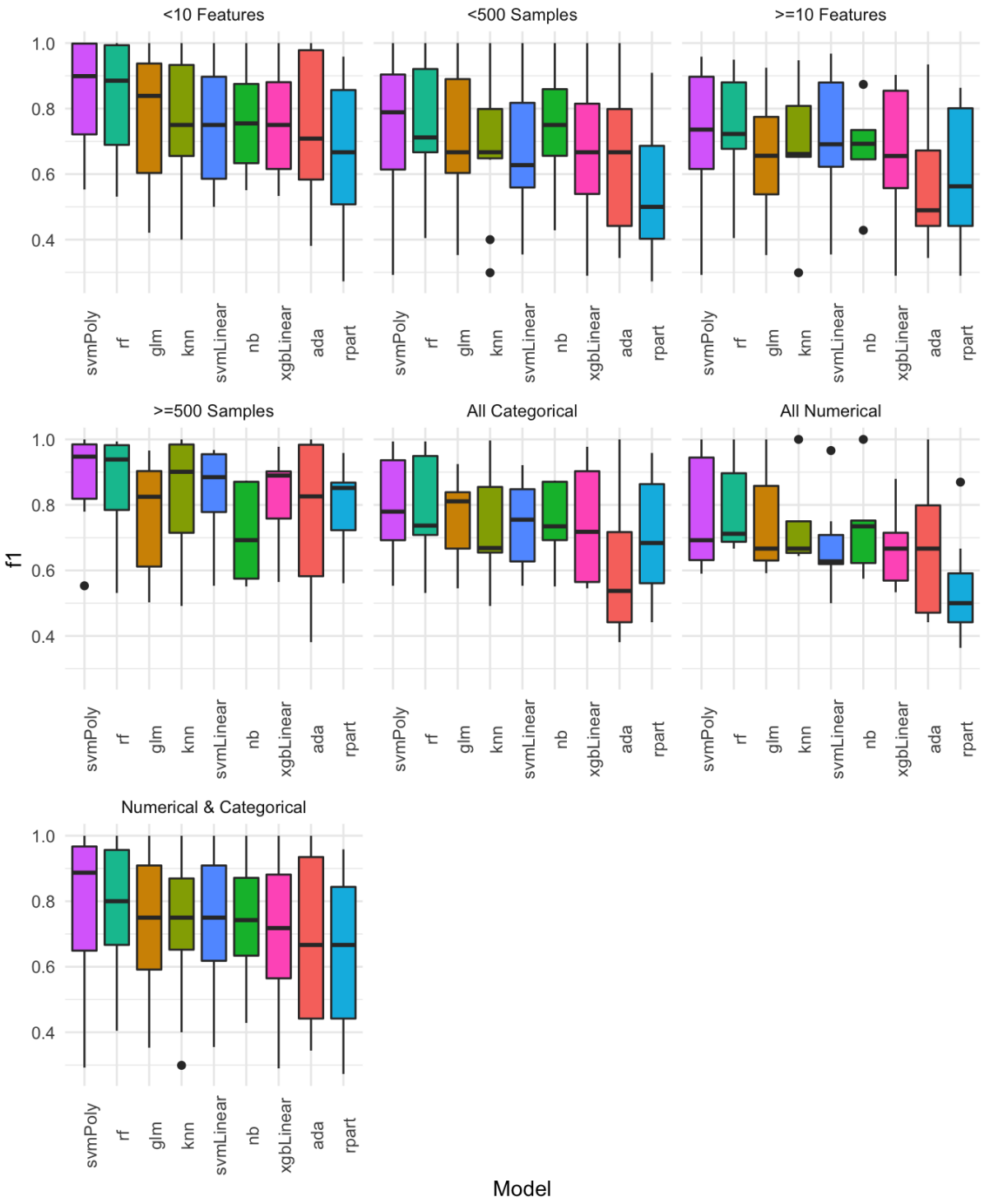

Figura 4.19: Compación de los distintos tipos de datasets con todos los modelos

Capítulo 5

Conclusión

Los resultados obtenidos concordaban en ocasiones con los esperados teóricamente. El modelo rpart al ser muy sencillo debería ser siempre uno de los que obtenga peores resultados. Por el contrario, el modelo rf ha demostrado funcionar bien en la mayoría de los escenarios. En cuanto a svmPoly y svmLinear, svmPoly siempre ha mostrado mejores resultados, lo cuál es de esperar siempre y cuando se evite el sobreajuste. Parece ser que el ajuste de hiperparámetros de svmPoly con cross-validation ha sido satisfactoria.

Sorprende el pobre resultado de xgbLinear y ada. En especial xgbLinear, ya que es el modelo que consigue los mejores resultados en la mayoría de competiciones de Kaggle. Estos pobres resultados pueden ser debidos a que los datasets no eran lo suficientemente complejos o a que requieren un ajuste de hiperparámetros más cuidadoso.

Los modelos de glm, k-nn y nb a pesar de ser sencillos han aparecido en posiciones intermedias en los ránkings por lo que pueden ser modelos a considerar en casos en los que el peso del modelo sea clave.

5.1. Asignaturas del máster

En la realización de este trabajo de fin de máster se han profundizado conceptos dados durante el curso de varias de las materias impartidas pero especialmente de las asignaturas de Minería de datos, Técnicas y métodos de ciencia de datos y Simulación y métodos de computación. Puesto que en las citadas asignaturas se han explicado los diferentes tipos de modelos estadísticos, codificación de los algoritmos R y su base teórica estadística.

En este trabajo se pueden encontrar conceptos de la asignatura de Técnicas y métodos de ciencia de datos concentrados sobre todo en los primeros dos módulos de la asignatura. Los conocimientos de los temas específicos estudiados en esta materia han sido utilizados en este trabajo se pueden resumir en programación científica en R, desarrollo de proyectos de ciencia de datos (fases que conlleva: limpieza de datos, procesamiento, modelado, evaluación y decisión), reproducibilidad, variables aleatorias, vectores aleatorios, modelos de distribución de probabilidad continuos y multivariantes.

Las asignaturas de Simulación y métodos de computación y Minería de datos fueron continuación una de la otra. De las asignaturas de Simulación y métodos de computación y Minería de datos se han usado los conceptos adquiridos sobre los modelos lineales generalizados (GLM) y SVM. Como se ha podido observar en este trabajo tres de los modelos seleccionados como objeto de estudio han sido precisamente GLM, SVM lineal y SVM polinomial. Además se han incorporado conceptos sobre aprendizaje de información como son modelos de boosting como ada y xgbLinear y además aquí también se cubrió el algoritmo de randomForest (rf). Durante este curso también se estudiaron algoritmos basados en aprendizaje por similitud como K-nn, que también se ha recogido en este trabajo.

También se han resultado de utilidad durante este trabajo técnicas vistas durante el curso de Visualización, comunicación y presentación de resultados. Los conceptos adquiridos durante esta asignatura han estado presentes a la hora de realizar los gráficos de comparación entre modelos. Se ha intentado que fueran lo más simples posibles para que lo más importante, la información, quedara representada de la manera más limpia. En mi opinión gráficos demasiado trabajados y que añaden diversas ”features”, pueden resultar muy vistosos, pero realmente distraen de la idea que realmente se quiere transmitir.

5.2. Nuevos conocimientos

Los conocimientos que se han adquirido tras la realización de este trabajo de fin de máster o que se han profundizado más allá de lo visto en clase han sido sobre todo tres: librería caret, paralelización en R y estimación de hiperparámetros mediante validación cruzada.

La librería caret se dio brevemente durante el máster, pero ha servido de base para la realización de este trabajo. Se ha profundizado en bastantes de las funciones que la librería permite

para la realización de proyectos de ciencia de datos, especialmente en el entreno de modelos y su evaluación. Una de las funciones que permite el paquete caret es la paralelización en R. Al contrario que en los casos de los conceptos anteriores, la paralelización en R no se dio durante el curso, siendo un nuevo concepto que se ha estudiado durante la realización de este trabajo. Caret cuenta con herramientas para reducir el tiempo del modelado de algoritmos como el argumento allowParallel en la función train(). Esta función, train(), y otras muchas del paquete caret (como rfe, sbf, bag, avNNet, etc, que no han sido estudiadas en este trabajo) recibieron un argumento adicional es sus respectivos archivos de control llamado allowParallel. Cuando este argumento recibe el valor de "True", el código se ejecutará en paralelo si tiene registrado un backend en paralelo, con al valor "False" el backend se ignora. Si se utiliza un backend paralelo con procesadores P, la combinación de estas funciones creará procesos $P\hat{2}$. Dado que algunas operaciones se benefician más de la paralelización que otras, gracias a caret, el usuario tiene la capacidad de concentrar recursos informáticos para funciones específicas.

Algo que también he aprendido en el transcurso de este TFM es el uso de cross-validation a la hora de estimar hiperparámetros. Dado que se deseaba un código general que pudiera funcionar con todo tipo de datasets y modelos, y dado que había datasets con pocas muestras, la designación de parte del set de datos exclusivamente para validación no era aceptable. Por ello, se usa cross-validation para poder obtener hiperparámetros que generalicen bien más allá de los datos de entrenamiento/validación sin requerir una gran cantidad de datos. Por motivos de tiempo de computación usamos cross-validation con k=2 para cada combinación de hiperparámetros. Se recomienda en futuros estudios usar un k mayor siempre se disponga de una capacidad de computación mayor.

5.3. Lineas de futuro

Este trabajo de fin de máster, como todos, se realizó durante un período de tiempo definido, por lo que obviamente tiene varias áreas de mejora que por tiempo no se han podido incorporar. Una de las áreas de futuro podría ser la de ampliar los tipos de conjuntos de datasets permitiendo que incluyan texto, y modelos para clasificación no binaria. Para que el código propuesto en este trabajo fuera lo más completo posible, otra línea de futuro sería la de añadir problemas de regresión y pudiendo especificar qué tipo de problema se quiere realizar y que el propio código

sepa qué tipo de modelos debe usar.

Además estaría bien añadir otras métricas para evaluar los modelos que sirvan de apoyo a las propuestas en este trabajo, para poder disponer de una información más detallada sobre qué tipo de modelo es más efectivo para el tipo de datos que necesitamos usar.

Otra propuesta para un futuro desarrollo de este trabajo es la descarga automática de los datos por métodos vistos durante el máster (librerías Scrapy de Python, por ejemplo). Incluyendo también una automatización para la creación del fichero metadata, que aporta al código información sobre cómo tratar (ETL) los datos a cargar.

Apéndice A

Tabla acrónimos

Acrónimo	Definición
svmPoly	Support Vector Machine with Polynomial kernel
svmLinear	Support Vector Machine with Linear kernel
ada	Boosted Classificaton Trees
glm	Generalized Linear Model
rf	Random Forest
rpart	Recursive Partitioning and regression Trees
knn	K-Nearest Neighbors
nb	Naive Bayes
xgbLinear	eXtreme Gradient Boosting
F1	F1 Score
ETL	Extract, Transform, Load
AI	Artificial Intelligence
FP	False Positives
TP	True Positives
FN	False Negatives
TN	True Negatives
ML	Machine Learning
ANN	Artificial Neural Network
UCI	University of California, Irvine

Cuadro A.1: Acrónimos

Bibliografía

[1] B. Auguie, A. Antonov, and M. B. Auguie. Package gridextra. *Miscellaneous Functions for Grid Graphics*, 2017.

[2] F. S. Caparrini, T. F. de Grado Dirigidos, T. F. de Máster Dirigidos, E.-S. NetLogo, I. Artificial, P. con NetLogo, D. Learning, A. Automático, M. Cultural, C. Natural, et al. Investigación: sistemas complejos. *Inteligencia Artificial*, 2020:21, 2016.

[3] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[4] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[5] K. D. Foote. A brief history of machine learning, 2019.

[6] J. Grau, I. Grosse, and J. Keilwagen. Prroc: computing and visualizing precision-recall and receiver operating characteristic curves in r. *Bioinformatics*, 31(15):2595–2597, 2015.

[7] G. Grolemund and H. Wickham. R for data science. 2018.

[8] O. Harrison. Machine learning basics with the k-nearest neighbors algorithm. *Towards Data Science. September*, 10, 2018.

[9] D. O. Hebb, J. Martinez, and S. Glickman. The organization of behavior-a neuropsychological theory-hebb, do, 1994.

[10] A. Hintze. Computers to humans: Shall we play a game?, 2017.

[11] D. Krstajic, L. J. Buturovic, D. E. Leahy, and S. Thomas. Cross-validation pitfalls when selecting and assessing regression and classification models. *Journal of cheminformatics*, 6(1):1–15, 2014.

[12] M. Kuhn. The caret package. *R Foundation for Statistical Computing, Vienna, Austria. URL https://cran. r-project. org/package= caret*, 2012.

[13] M. Kuhn. Variable importance using the caret package. *Journal of Statistical Software*, 2012.

[14] M. Kuhn. A short introduction to the caret package. *R Found Stat Comput*, 1, 2015.

[15] M. Kuhn, J. Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, R. C. Team, et al. Package caret. *The R Journal*, 2020.

[16] C. Leach. Introduction to parallel computing in r. *Online tutorial, http://michaeljkoontz. weebly. com/uploads/1/9/9/4/19940979/parallel. pdf*, 2014.

[17] G. Louppe. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*, 2014.

[18] U. Malik. Cross validation and grid search for model selection in python.

[19] K. P. Murphy et al. Naive bayes classifiers. *University of British Columbia*, 18:60, 2006.

[20] Naveen. Data visualization in r.

[21] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda. The cart decision tree for mining data streams. *Information Sciences*, 266:1–15, 2014.

[22] R. E. Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[23] T. Sing, O. Sander, N. Beerenwinkel, and T. Lengauer. Rocr: visualizing classifier performance in r. *Bioinformatics*, 21(20):3940–3941, 2005.

[24] I. Steinwart and A. Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[25] I. Stephen. Perceptron-based learning algorithms. *IEEE Transactions on neural networks*, 50(2):179, 1990.

[26] W. W. Stroup. *Generalized linear mixed models: modern concepts, methods and applications*. CRC press, 2012.

[27] UCI. Machine learning repository.

[28] H. Wickham. ggplot2. *Wiley Interdisciplinary Reviews: Computational Statistics*, 3(2):180–185, 2011.

[29] H. Wickham and J. Bryan. readxl: Read excel files. r package version 1.0. 0. *URL https://CRAN. R-project. org/package= readxl*, 2017.

[30] H. Wickham and M. H. Wickham. Package stringr. 2019.