Índice general

Capítulo 1

Modelización

A continuación se va a utilizar el conjunto de datos construido en el capítulo [Capítulo 3 sobre la construcción del conjunto de datos][Construcción del conjunto de datos] para entrenar los modelos de clasificación binaria explicados en la [Sección 2.3][Modelos]. Es evidente que el rendimiento de los modelos debe evaluarse en observaciones futuras, por lo que las técnicas habituales de validación cruzada o partición aleatoria en entrenamiento-test no son adecuadas para este problema, ya que sufrirían el llamado efecto *look-ahead*. Por tanto, el enfoque que se seguirá en este trabajo será trabajar con una partición en entrenamiento-validación-test construida a partir de la ordenación temporal de las observaciones.

Se compararán los resultados obtenidos en 7 modelos diferentes: Regresión Logística con penalización, Regresión Logística con penalización usando PCA, Árboles de Decisión, Bosques Aleatorios, KNN, SVM lineal y SVM radial.

Se ha seguido el flujo de trabajo habitual del paquete *tidymodels*:

1º. Crear una partición temporal en entrenamiento (60 %), validación (20 %) y test (20 %), que será utilizada en todos los modelos.

2º. Definir cada uno de los modelos, indicando los parámetros del modelo que deberán ajustarse.

3º. Crear la receta (*recipe*) con el preprocesamiento que se usará en cada modelo (lo que se conoce como **ingeniería de características**). Como se observó en el [Capítulo 4 sobre análisis exploratorio de datos][Análisis exploratorio de datos], se incluirán en todos los modelos variables categóricas que indiquen el día de la semana y el mes de cada observación, haciendo uso de la función `step_date`. Igualmente, como también se indicó en el EDA, se modificará la variable *uso_suelo* para unificar todos los niveles que no sean agrícolas o forestales en un solo nivel que se llamará *Otro*. Esto último se hará fuera del *workflow*, antes de realizar la partición del conjunto de datos, haciendo uso de la función `fct_lump` del paquete *forcats* [1]. Los detalles del preprocesamiento que se ha llevado a cabo en cada modelo pueden consultarse en el código, que se adjunta en el [Apéndice B de código][Apéndice: Código] , donde aparecen debidamente comentados.

4º. Crear el *workflow* con el modelo y la receta.

5º. Crear la rejilla (*grid*) con los posibles valores de los parámetros que se deben ajustar.

6º. Entrenar el modelo para cada combinación de los valores de los parámetros a ajustar sobre los datos de entrenamiento.

7º. Evaluar el rendimiento de cada modelo sobre los datos de validación y seleccionar el mejor en base a las medidas de rendimiento ya mencionadas. El objetivo con estos modelos es predecir incendios forestales, por lo que es de vital importancia que los modelos funcionen especialmente bien en la clase positiva, es decir, que si un incendio se va a producir, que el modelo lo detecte. Sin embargo, es fundamental que el modelo tenga un buen desempeño general (un modelo que todo lo clasifique como incendio no serviría de nada, poniendo un ejemplo extremo). Por tanto, cada modelo se valorará de forma individual, considerando todas las métricas de rendimiento mencionadas y priorizando la sensibilidad (o *recall*). Sin embargo, en la mayoría de los casos maximizar la tasa de acierto maximiza también la sensibilidad, garantizando además un buen desempeño general. Por ello, en la mayoría de modelos se maximizará la tasa de acierto, pero porque analizando las salidas individualmente se ha considerado que es la mejor opción ya que produce la mayor sensibilidad sin bajar demasiado las otras medias.

1.1. Regresión Logística con penalización

Antes de aplicar este modelo, se han transformado las variables categóricas a variables *dummy* y se han tipificado todas las variables (media 0 y varianza 1). Los parámetros a ajustar son λ (parámetro de penalización o *penalty*) y α (parámetro de mezcla o *mixture*). Se consideran 10 valores equiespaciados para cada parámetro (en el caso de λ entre 10^{-4} y 10^{-1} y el caso de α entre 0 y 1) y se construye el *grid* tomando todas las combinaciones de estos valores.

Las métricas obtenidas por cada combinación de parámetro sobre los datos de validación se representan en la Figura 1.1.

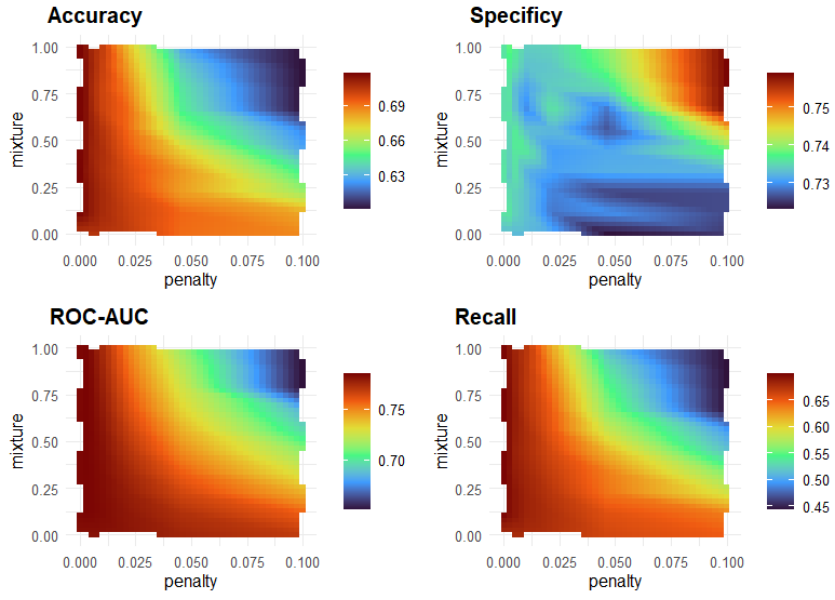

Figura 1.1: Métricas de rendimiento de los modelos de Regresión Logística con penalización. *Fuente: Elaboración propia.*

Finalmente, se elige el modelo maximiza la tasa de acierto, cuyos parámetros son: $\alpha = 1$ y $\lambda = 0.000464$. Es decir, un modelo de Regresión Logística *lasso* puro. Los coeficientes de

este modelo se muestran en la Figura **??** del [Apéndice A.2. Salidas de los modelos][Salidas de los modelos].

1.2. Regresión Logística con penalización usando PCA

A continuación se considera el mismo modelo de Regresión Logística con penalización que en la sección, anterior pero en lugar de trabajar con los datos directamente, se aplica análisis de componentes principales sobre los datos normalizados en el preprocesamiento, ajustando el número de componentes principales utilizadas. Para construir el *grid* de parámetros se consideran los mismos valores que en el modelo sin PCA para los parámetros de penalización y mezcla pero ahora se consideran también 7 posibles valores para el número de componentes principales ($\{20, 25, 30, ..., 50\}$). Nótese que anteriormente, cuando se realizó PCA en la [Sección 4.3][Análisis multivariantes de las variables numéricas] tan solo se consideraron las 18 variables numéricas del conjunto de datos antes de aplicar ingeniería de características, y se concluyó que eran necesarias al menos 14 componentes principales para explicar el 90 % de la varianza de los datos. En cambio, aquí se están considerando todas las variables después de aplicar ingeniería de características, por lo que se están incluyendo todas las variables *dummy* creadas a partir de los factores. Esto supone un total de 59 variables (véase la Figura **??** del [Apéndice A.2][Salidas de los modelos], donde aparecen todas las variables del modelo una vez aplicada ingeniería de características), por lo que es de esperar que el número de componentes principales necesarias para explicar un porcentaje alto de la varianza de los datos aumente sensiblemente con respecto a los resultados obtenidos considerando tan solo las variables numéricas. Es por ello que se consideran al menos 20 componentes principales en el ajuste.

Finalmente, el modelo que maximiza la tasa de acierto es el que tiene 40 componentes principales, $\alpha = 0.333$ y $\lambda = 0.00464$.

1.3. Árboles de Decisión

Se construirán los Árboles de Decisión usando el índice de Gini como medida de impureza utilizada para determinar el par variable-corte en cada nodo, y se elegirá el parámetro de coste-complejidad (α) que maximice la tasa de acierto. Tras valorar también el uso de la Entropía como medida de impureza y realizar algunas pruebas previas, finalmente se optó por usar el índice de Gini, ya que viene implementado en los paquetes considerados y que, en las pruebas realizadas, el uso de una medida u otra no supuso cambios significativos en los resultados obtenidos. Se considera un *grid* con 10 valores del parámetro de coste-complejidad que oscilan entre $1.28e-10$ y $3.02e-2$. La mejor tasa de acierto en el conjunto de validación se obtiene con $\alpha = 0.00182$. En la Figura 1.2 se muestran las distintas métricas de rendimiento sobre los datos de validación para cada uno de los valores del parámetro a ajustar.

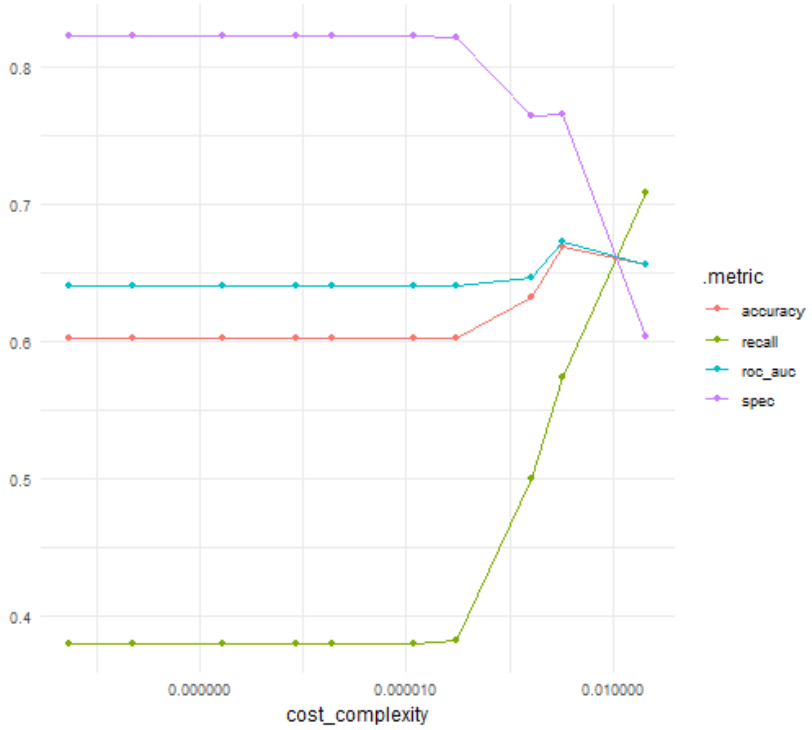

Figura 1.2: Métricas de rendimiento del Árbol de Decisión en función del parámetro de coste-complejidad. *Fuente: Elaboración propia.*

1.4. Bosques Aleatorios

En este modelo se han ajustado los parámetros *mtry* (el número de variables que se seleccionarán aleatoriamente en cada nodo) y *min_n* (el número de observaciones en un nodo a partir del cual no se sigue dividiendo y se convierte en nodo hoja). En este caso se ha optado por un enfoque diferente, motivado por el amplio rango de valores que puede tomar el parámetro *min_n* y por las limitaciones computacionales del equipo disponible. Tras realizar varias pruebas se ha observado que, con estos datos, al ajustar el parámetro *trees* (el número de árboles a considerar) no se obtiene una mejorar significativa de los resultados frente a considerar el valor por defecto de 1000 árboles, por lo que se ha preferido dejar este parámetro fijo.

De esta forma, la estimación de parámetros se ha hecho en dos etapas. En una primera etapa se ha fijado el parámetro $mtry = 4$ y se ha estimado el parámetro *min_n* considerando para ello un *grid* equiespaciado de 1000 a 2500 tomando valores de 100 en 100 ($\{1000, 1100, ..., 2500\}$). Para elegir entre los distintos modelos, esta vez se ha usado como criterio la sensibilidad, obteniendo el valor más elevado para $min\_n = 2100$. En la segunda etapa, una vez *min_n*, este se ha considerado fijo y se ha estimado *mtry*, considerando una rejilla de 10 valores equiespaciados tomados del 1 al 10 ($\{1, 2, ..., 10\}$). De nuevo se ha utilizado la sensibilidad para elegir el modelo final, eligiendo así $min\_n = 7$. En la Figura 1.3 se recogen los resultados de las dos etapas de *tuning*. El modelo final elegido tiene $min\_n = 2100$ y $mtry = 7$.

Aquí es necesario hacer un inciso sobre el motivo de considerar valores de *min_n* tan elevados, en especial si se tiene en cuenta que la muestra con la que se trabaja está formada por 20.000 registros. En un inicio, se trató de ajustar ambos parámetros a la

vez, considerando valores usuales para el parámetro min_n (del orden de la decena). Sin embargo, los tiempos de entrenamiento eran sumamente largos y el rendimiento de los modelos era pésimo. Si bien el área bajo la curva ROC y la tasa global de acierto eran aceptables y la tasa de acierto en la clase negativa era excelente, en la clase positiva las tasas de acierto obtenidas era inferiores a 0.5, indicando que los modelos no estaban funcionando adecuadamente. Entonces, se comenzaron a realizar pruebas con valores más elevados de *min_n*, hasta conseguir un rendimiento aceptable en la clase positiva, que es lo principal en este problema. De esta forma, se llegó a los valores del parámetro *min_n* que se presentan en el párrafo anterior, que llevan a considerar árboles poco profundos. Sería interesante, en trabajos futuros, seguir explorando vías de mejora de este modelo, tratando de desentrañar las causas del rendimiento poco prometedor obtenido con valores de *min_n* moderados.

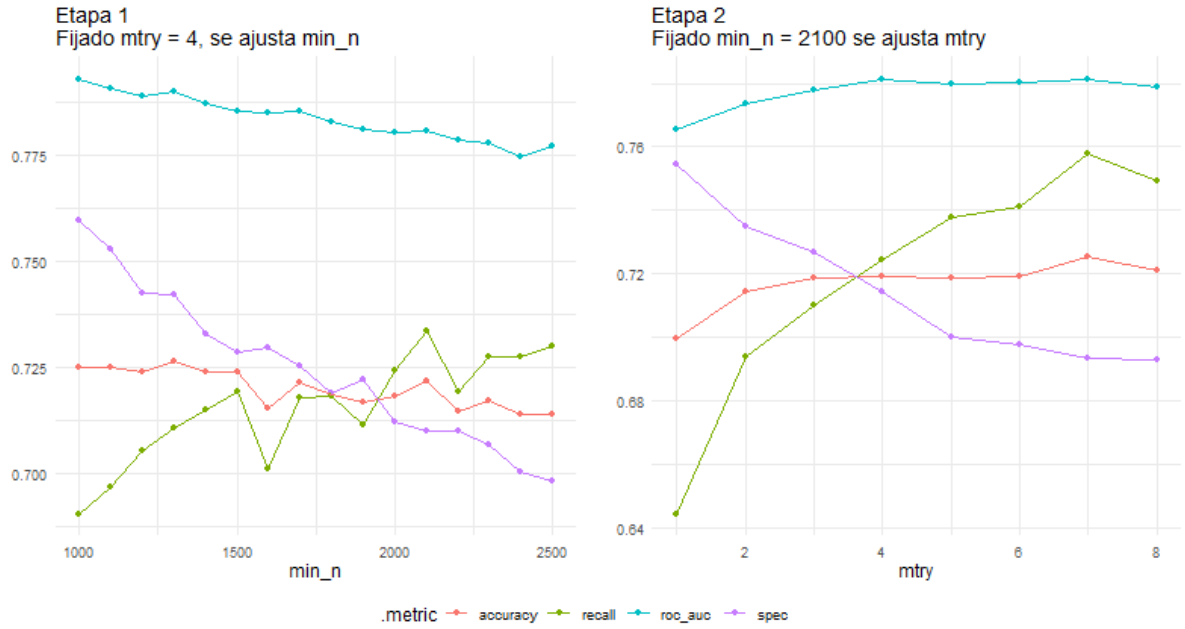

Figura 1.3: Métricas de rendimiento de *Random Forest* en función de los parámetros. *Fuente: Elaboración propia.*

1.5. KNN

Para aplicar el modelo, primero se han transformado las variables categóricas en variables *dummy* y, posteriormente, se han tipificado todas las variables. Se ha usado la distancia euclídea entre los vectores transformados. Para ajustar el parámetro k del modelo se han tomado valores entre 1 y 400. La mayor tasa de acierto sobre los datos de validación se ha obtenido con $k = 275$. Los resultados del *tuning* se muestran en la Figura 1.4.

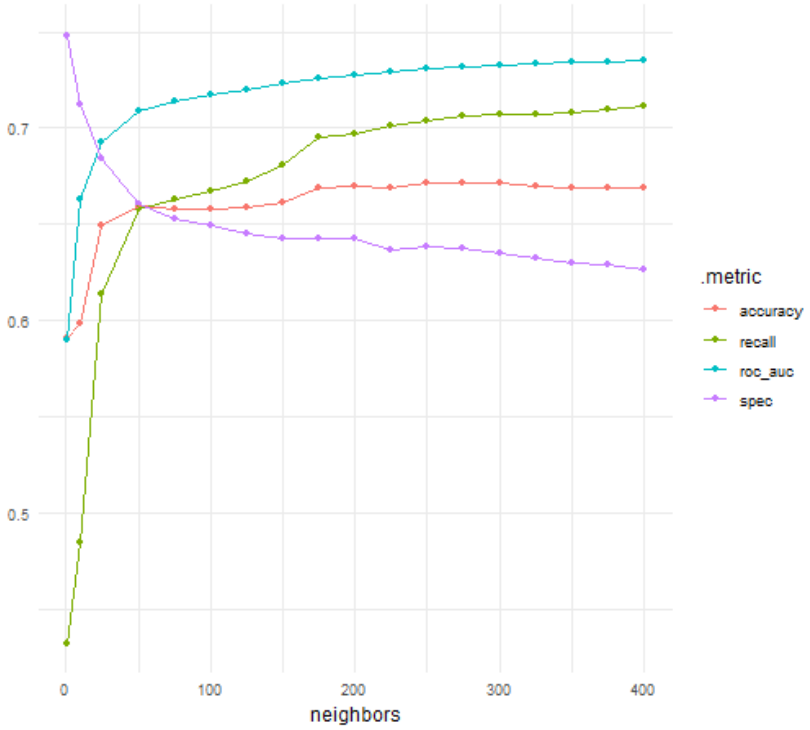

Figura 1.4: Métricas de rendimiento de KNN en función del número de vecinos. *Fuente: Elaboración propia.*

1.6. SVM lineal

Antes de construir el modelo, se han transformado las variables categóricas usando variables *dummy* y se han tipificado todas las variables. Se ha probado con 15 valores del parámetro *coste* entre 0.001949 y 24.666648. La mayor tasa de acierto y el mayor *recall* se han obtenido para $C = 0.0437$. Los resultados del *tuning* se muestran en la Figura 1.5.

1.7. SVM radial

Por último, se ha construido el modelo de SVM usando un *kernel* gaussiano, con la intención de comprobar si el uso de esta función kernel es capaz de mejorar la separabilidad de los datos. Se ha considerado este *kernel* y no otro por ser el de uso más común. El preprocesamiento ha sido el mismo que en el caso del *kernel* lineal. Dado el elevado tiempo de entrenamiento de este modelo solo se ha probado con 8 combinaciones de valores para los parámetros C y γ, que oscilan entre 0.005 y 31.7 y entre 0 y 0.05, respectivamente. La mayor tasa de acierto sobre los datos de validación se ha conseguido para $C = 31.7$ y $\gamma = 0.0000496$.

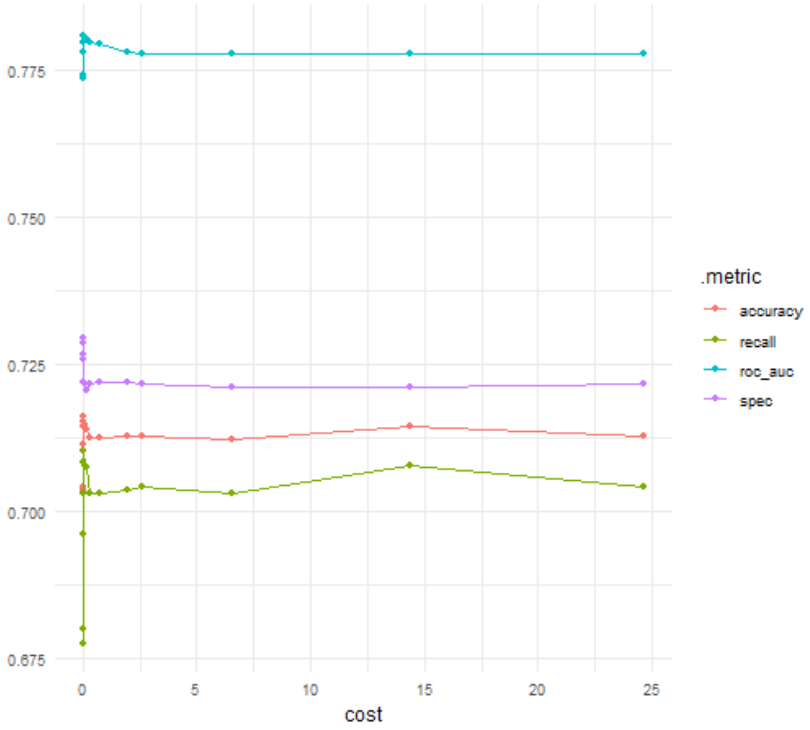

Figura 1.5: Métricas de rendimiento de SVM en función del parámetro C. *Fuente: Elaboración propia.*

1.8. Evaluación y comparación de modelos

1.8.1. Comparativa sobre el conjunto de validación

A continuación se muestran las métricas de cada uno de los modelos seleccionados en los datos de validación en la Tabla 1.1 y en la Figura 1.6. Las curvas ROC de todos los modelos se muestran en la Figura 1.7.

Puede observarse que los resultados obtenidos por todos los modelos son bastante similares. Destacan el modelo de Bosque Aleatorio y el de Regresión Logística con penalización, el primero por ser el que tiene la tasa de acierto y la sensibilidad más elevadas, y el segundo por dar los mejores resultado en cuanto a precisión y especificidad. Las curvas ROC de los modelos de bosque aleatorio, regresión logística con penalización, SVM lineal y SVM radial son prácticamente iguales. Los modelos más pobres son la regresión logística aplicando PCA, KNN y el árbol de decisión. Seguramente, esto sea debido a que son modelos demasiado simples dada la complejidad del problema al que se quieren aplicar.

model_name	roc_auc	accuracy	recall	specificity	precision
lr	0.785	0.719	0.700	0.738	0.727
lr_pca	0.727	0.659	0.624	0.694	0.670
dt	0.656	0.656	0.709	0.604	0.641
rf	0.781	0.725	0.758	0.693	0.711
svm_linear	0.781	0.716	0.710	0.722	0.718
svm_rbf	0.777	0.710	0.692	0.728	0.717
knn	0.732	0.672	0.706	0.637	0.660

Tabla 1.1: Métricas de los modelos seleccionados sobre el conjunto de validación. *Fuente: Elaboración propia.*

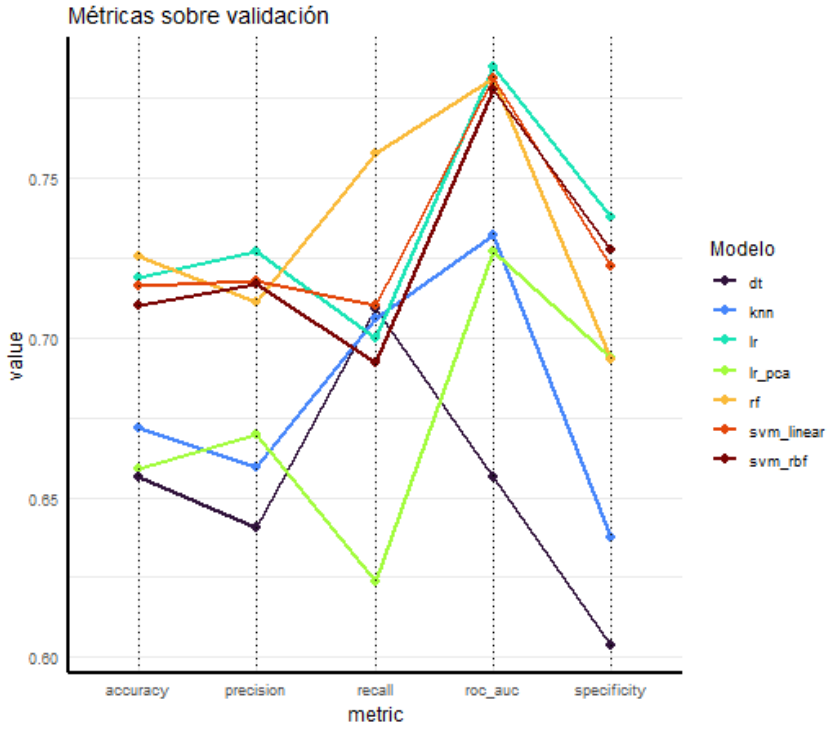

Figura 1.6: Gráfico de métricas obtenidas sobre el conjunto de validación por cada uno de los modelos seleccionados. *Fuente: Elaboración propia.*

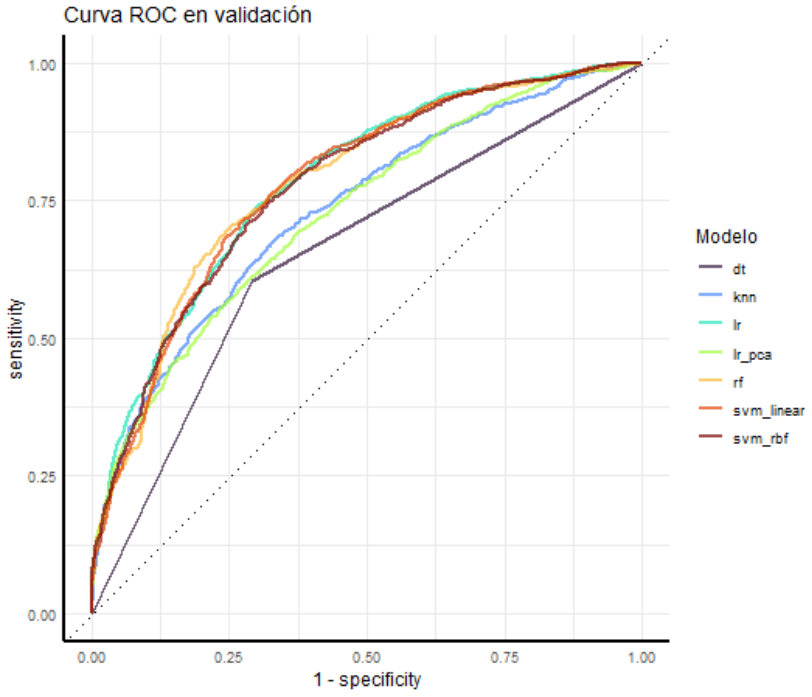

Figura 1.7: Curvas ROC sobre el conjunto de validación. *Fuente: Elaboración propia.*

A continuación, se incluye un fragmento del código en el que se ilustra el uso de funciones propias del flujo de trabajo habitual de *tidymodels* dentro de la filosofía *"tidy data"* propia del ecosistema *tidyverse*, haciendo uso de estructuras anidadas y funciones de orden

superior. En este fragmento, se selecciona la mejor combinación de parámetros para cada modelo, usando la tasa de acierto global como criterio; a continuación, se obtienen todas las medidas de rendimiento sobre el conjunto de validación para cada modelo ajustado, haciendo uso de la función propia `get_metrics`; y por último, se crea la curva ROC de cada modelo ajustado sobre el conjunto de validación. El objeto `models_tune` contiene los resultados del ajuste de cada combinación de parámetros considerada para cada modelo sobre el conjunto de validación.

```
models = models %>%
  mutate(best_tuning = map(models_tune,
                           function(x) select_best(x,
                                                   metric = "accuracy")),
         best_metrics = map2(models_tune,
                             best_tuning,
                             ~ collect_predictions(.x,
                                                   parameters = .y) %>%
                               get_metrics() %>%
                               extract2(1)),
         roc = map2(models_tune,
                    best_tuning,
                    ~ collect_predictions(.x,
                                          parameters = .y) %>%
                      roc_curve(fire, .pred_0))
         )
```

1.9. Comparativa sobre el cojunto test

Por último, para conocer la capacidad de generalización de los modelos construidos, estos se evaluarán sobre nuevas observaciones, el conjunto de datos test. Recuérdese que el entrenamiento de los modelos se ha realizado con el conjunto de entrenamiento, formado por 12.927 observaciones tomadas entre el 2002 y mediados de 2014, y para ajustar los parámetros de cada modelo, se han utilizado 4.309 observaciones tomadas entre mediados de 2014 y mediados de 2019. Finalmente, se evaluará la capacidad de predicción de los modelos sobre 4.310 nuevas observaciones tomadas entre mediados de 2019 y 2022. Para ello, primero se unirán los conjuntos de entrenamiento y validación para reentrenar los modelos con la configuración de parámetros seleccionada en cada caso, y posteriormente se compararán los valores predichos por los modelos con los valores reales. Los resultados obtenidos se muestran en la Tabla 1.2 y el la Figura 1.8. Las curvas ROC de los distintos modelos sobre el conjunto de datos test se muestra en la Figura 1.9.

En este caso, los mejores resultados en todas las medidas los da el modelo de Regresión Logística con penalización. Los modelos de SVM muestran resultados bastante similares entre ellos y prácticamente iguales al modelo de Regresión Logística. Sobre los datos test, el modelo de Bosque Aleatorio ha dado un rendimiento peor que el obtenido en validación, quedando por detrás de los tres modelos ya comentados, aunque la sensibilidad de todos estos modelos es prácticamente igual. De nuevo, los peores resultados los dan los modelos de Regresión Logística aplicando PCA y el Árbol de Decisión, seguidos del KNN.

model_name	roc_auc	accuracy	recall	specificity	precision
lr	0.795	0.710	0.726	0.693	0.718
lr_pca	0.715	0.645	0.631	0.661	0.667
dt	0.667	0.668	0.712	0.621	0.670
rf	0.762	0.699	0.727	0.669	0.703
svm_linear	0.790	0.705	0.729	0.680	0.710
svm_rbf	0.789	0.706	0.727	0.683	0.712
knn	0.744	0.673	0.719	0.624	0.673

Tabla 1.2: Métricas sobre el conjunto test. *Fuente: Elaboración propia.*

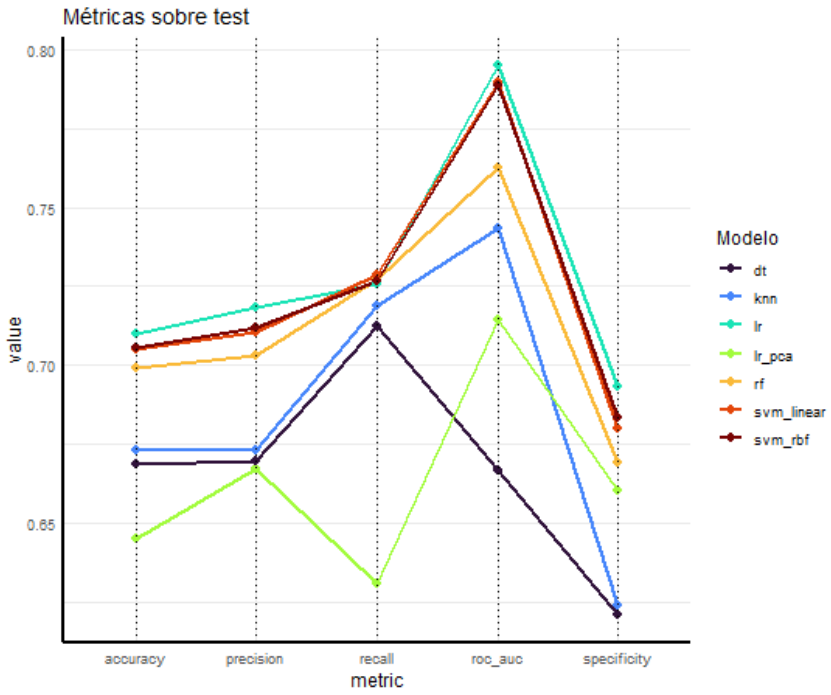

Figura 1.8: Métricas obtenidas sobre el conjunto test por cada uno de los modelos seleccionados. *Fuente: Elaboración propia.*

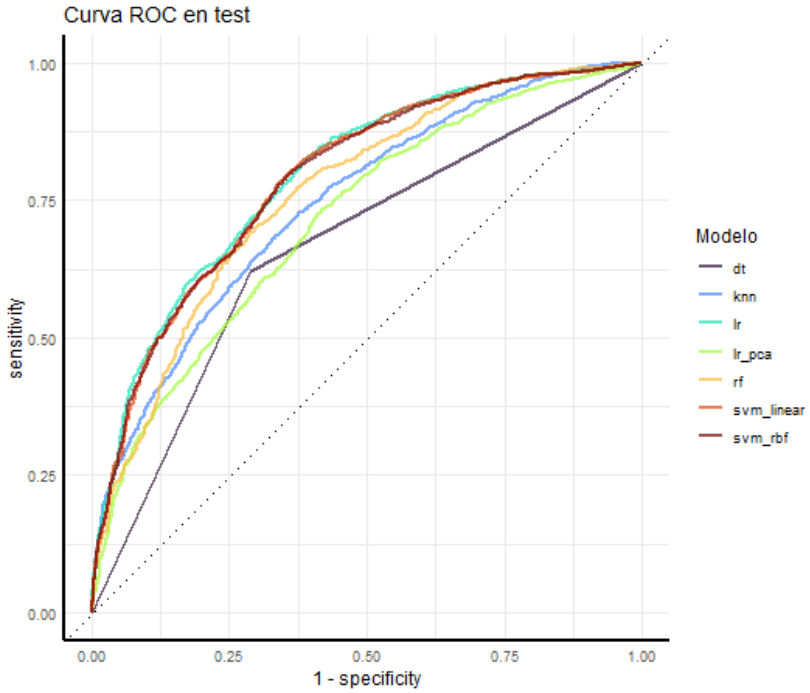

Figura 1.9: Curvas ROC sobre test. *Fuente: Elaboración propia.*

En el fragmento de código que se incluye a continuación, se realizan las siguientes operaciones sobre todos los modelos considerados:

1. Se finaliza el entrenamiento de los modelos, usando la combinación de parámetros previamente seleccionada para cada uno.
2. Se reentrenan sobre la unión de los conjuntos de entrenamiento y validación.
3. Se obtienen las medidas de rendimiento sobre el conjunto test.
4. Se construye la curva ROC de cada modelo sobre el conjunto test

```
models = models %>%
  mutate(final_workflow = map2(models_workflow,
                               best_tuning,
                               finalize_workflow),
         last_fit = map(final_workflow,
                        function(x) last_fit(x,
                                             splits,
                                             add_validation_set=T)),
         test_metrics = map(last_fit,
                            ~collect_predictions(.x) %>%
                              get_metrics() %>%
                              extract2(1)),
         test_roc = map(last_fit,
                        ~collect_predictions(.x) %>%
                          roc_curve(fire, .pred_0))
         )
```

La fácil compresión del código, la posibilidad de usar funciones de orden superior que permiten realizar operaciones a múltiples elementos de forma sencilla y eficiente, y el uso

de estructuras anidadas y funciones de orden superior que simplifican la programación, muestran la potencia de la integración de *tidymodels* dentro del universo *tidyverse*.

En el siguiente capítulo, se evaluará el rendimiento de los modelos construidos en esta sección al ser aplicados a dos casos prácticos. De esta forma, se podrá valorar la utilidad de estos en la realidad y conocer sus limitaciones.

Bibliografía

[1] Wickham, Hadley (2023). *forcats: Tools for Working with Categorical Variables (Factors)*.
https://CRAN.R-project.org/package=forcats. R package version 1.0.0.