# Aprendizaje Automático - Modelos probabilísticos y ensembles

**Ciencia de Datos e IA - 3º**
Gonzalo Lope Carrasco - g.lope@alumnos.upm.es
Javier Pérez Vargas - javier.perez.vargas@alumnos.upm.es
Mario Ruiz Vaquett - mario.ruiz@alumnos.upm.es

Abril 2024

# Índice

# 1. Introducción

En esta práctica tenemos un conjunto de datos *train* y un conjunto *test* que constan, respectivamente, de 3000 y 2000 instancias. Además, el número de variables es 90, aunque no parecen tener ninguna semántica clara o específica. Se trata de un problema de aprendizaje supervisado, concretamente de clasificación, donde la clase a predecir (variable **class**) tiene 3 posibles valores. Esta es una muestra de los datos.

| **lonsb** | **qpzkm** | **knyrk** | **...** | **pswtk** | **onmrn** | **pxlhg** | **class** |
|---|---|---|---|---|---|---|---|
| -2.724062 | 26.536867 | -0.075150 | ... | -1.193564 | 7.250148 | 10.248741 | 2 |
| -0.690392 | -5.542998 | -0.523969 | ... | -0.772035 | 1.944000 | -0.499702 | 0 |
| -0.265749 | -10.673678 | -1.002556 | ... | -0.652022 | 0.984688 | 9.409884 | 2 |

Cuadro 1: Tabla de datos reducida con algunas variables seleccionadas.

El objetivo será utilizar los modelos **probabilísticos** aprendidos durante el curso: *Naive Bayes*, *Augmented Naive Bayes* y *Análisis Discriminante*; así como los modelos de ***ensembles***: *Bagging*, *Random Forest*, *AdaBoost* y *Gradient Boosting*. Veremos cómo funciona cada uno y qué ventajas y desventajas tienen.

# 2. Métodos

## 2.1. Métodos probabilísticos

### 2.1.1. Naive Bayes

Naive Bayes es un algoritmo de aprendizaje automático supervisado que se basa en el teorema de Bayes para la clasificación, asumiendo independencia condicional entre las características, lo que lo hace rápido y eficiente.

Existen varios algoritmos que implementan Naive Bayes, con la diferencia de la distribución que asumen para las variables. Como podemos observar en la tabla, todas las características son continuas, por lo que podemos asumir que siguen una distribución normal. Es decir, utilizaremos la variante **Gaussian Naive Bayes**. Utilizar cualquier otra distribución carecería de sentido. Usando todos los datos obtenemos una precisión del 66,6 % en el conjunto de test.

Como ya hemos mencionado, Naive Bayes asume que todas las características son independientes. Es por ello que puede empeorar el rendimiento cuando existen características dependientes. Después de hacer un estudio de la correlación entre variables hemos observado que estas 3 están muy relacionadas.

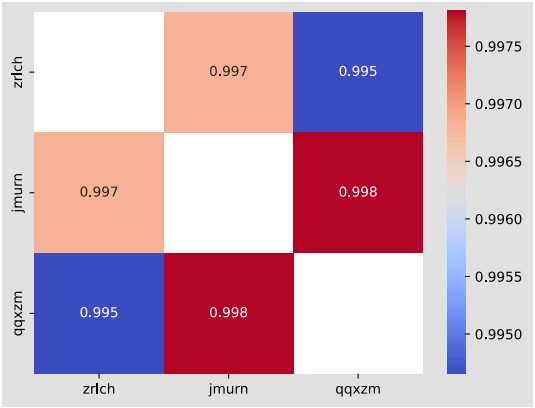


Figura 1: Correlaciones mayores a 0.5

Vemos que las 3 variables son casi totalmente dependientes entre sí, por lo que podrían perjudicar al modelo. Es por ello que hemos eliminado dos de las tres variables para hacer una segunda prueba y ver si mejoran los resultados. Eliminando esas dos variables obtenemos una precisión de 67.2 % en el conjunto de test.

Eliminar características redundantes es una buena práctica cuando trabajamos con Naive Bayes porque sino se rompe totalmente la asunción. Además, si eliminamos todas las variables correlacionadas entre sí, de al menos 0.1, obtenemos 67,5 % de precisión en test.

Viendo la mejora en precisión al quitar variables, decidimos probar con métodos de selección de variables para ver si se puede mejorar el resultado. Tras eliminar 45 variables (la mitad), haciendo uso de *SequentialFeatureSelector* con dirección 'hacia atrás' obtenemos 70.35 % en el conjunto de test. Con la selección 'hacia delante' (añadiendo variables en vez de quitando) y estipulando que el seleccionador pare únicamente cuando la precisión mejore menos de un 0.001 (alrededor de 3 nuevos datos bien clasificados), nos quedamos con 6 variables y obtenemos una precisión de 71.75 % en test.

Sabemos que Naive Bayes asume independencia entre variables, y por ello las variables irrelevantes no deberían afectar al modelo. No obstante, este último resultado parece hacer ver que, al menos en este dataset, eliminar características poco informativas es beneficioso para obtener modelos con mayor poder predictivo.

### 2.1.2. Augmented Naive Bayes

La estructura de Naive Bayes se puede representar como una sencilla red bayesiana, donde las diferentes variables se dibujan como nodos y una flecha entre ellas indica dependencia. En nuestro caso, sería algo así:

Figura 2: Naive Bayes como una red bayesiana

Como es lógico, no se llega a apreciar nada, pues Naive Bayes asume la independencia de todas las variables, de forma que todos los nodos (variables) son hermanos. No obstante, existe una variante, denominada **Tree-Augmented Naive Bayes**, que relaja la independencia condicional permitiendo un árbol de dependencias.

Utilizando este modelo y representándolo como una red bayesiana obtendríamos lo siguiente:

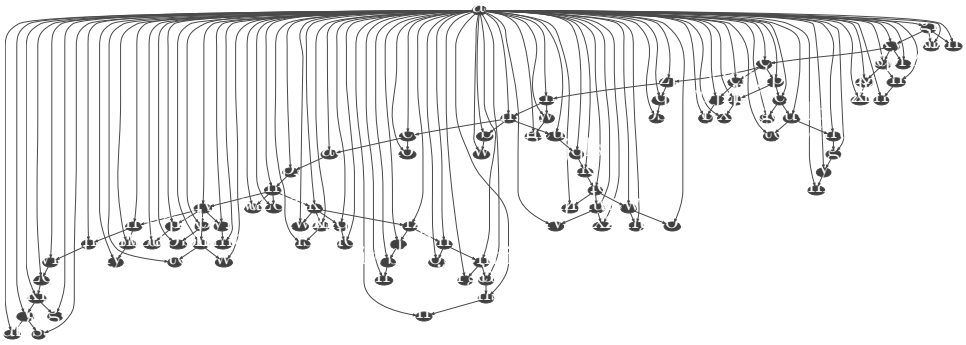

Figura 3: Tree-Augmented Naive Bayes como una red bayesiana

Las variable son ilegibles, pero lo importante es ver que se han permitido formar una serie de dependencias basadas en los datos, y podrían dar mejores resultados. Este primer modelo de red bayesiana proporciona una precisión de 58.75 % en el conjunto de test.

Probando con otros parámetros, específicamente aumentando el *priorWeight* (el peso de la información a priori) y modificando el *scoringType*, hemos llegado a otras estructuras de árbol donde se restringen más las dependencias y se obtienen mejores resultados, llegando a un 62.5 %.

Figura 4: Tree-Augmented Naive Bayes mejorado como una red bayesiana

Como vemos existen dependencias, pero en mucha menor escala que el anterior modelo. No obstante, lo que sí hemos observado es que en todas las redes bayesianas dibujadas aparecen relacionadas las 3 variables dependientes ya mencionadas.

### 2.1.3. Análisis Discriminante

En el análisis discriminante, se asume que cada clase de datos se puede modelar como una distribución normal. Su objetivo es encontrar una combinación lineal de características que mejor separe las clases.

Existen dos métodos: análisis discriminante **lineal** (LDA) y análisis discriminante **cuadrático** (QDA). El primero asume que las diferentes clases tienen la misma matriz de covarianza, es decir, asume que todas las clases tienen la misma forma y orientación. En cambio, el segundo permite que cada clase tenga su propia matriz de covarianza. Así, el QDA es mucho más flexible, pero al mismo tiempo más costoso, especialmente cuando hay muchas clases y muchas variables. Además, esta flexibilidad puede producir sobreajuste a los datos de entrenamiento.

Como resultado de esto, las fronteras de decisión en LDA serán lineales, mientras que en QDA serán curvas cuadráticas.

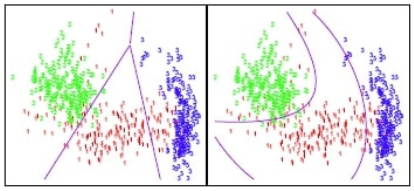

Figura 5: Ejemplo **con otro dataset**. LDA vs QDA

Hemos decidido usar ambos modelos para verificar cuál funciona mejor. Además de ver el rendimiento, nos podremos hacer una idea de cómo se distribuyen los datos en cada clase.

En primer lugar utilizamos el Análisis Discriminante Lineal. Tras probar varios *solver* y *shrinkage*, el mejor modelo obtenido (utilizando todas las variables) tiene una precisión de 40.55 % sobre el conjunto de test, lo cual nos sorprendió. Por ello, decidimos hacer selección de variables tal y como hicimos en Naive Bayes, para ver si podíamos obtener mejores resultados. Con la selección 'hacia delante' y el mismo umbral que antes, se obtiene un modelo con 2 variables, que también daba un resultado muy pobre. Lo más seguro es que LDA no funcione bien en este conjunto de datos por las suposiciones que hace, que ya comentaremos posteriormente.

El siguiente paso era probar el Análisis Cuadrático Discriminante. En este caso, sin modificar el conjunto de test, se obtenía una precisión muy baja, 48.35 %. Pero, para nuestra sorpresa, al realizar selección de variables, de la misma forma que con LDA, aumentaba la precisión en el conjunto de test hasta un 80.75 %, quedándose tan solo con 3 variables. Además, para este último modelo, decidimos ver cómo eran las matrices de covarianza de cada clase:

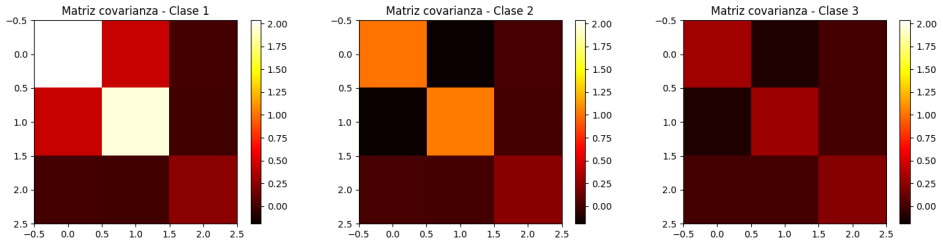


Figura 6: Matrices de covarianza de cada clase

Podemos observar que QDA permite que cada clase tenga su propia matriz de covarianza, y en este caso son bastante diferentes. Es por ello, seguramente, que QDA pueda sacar ventaja sobre LDA en este tipo de datasets, pues permite una mayor flexibilidad sobre la distribución de las clases.

## 2.2. Ensembles

### 2.2.1. Bagging

Bagging es un método de ensemble en el que se entrenan múltiples modelos utilizando conjuntos de datos *bootstrap* (muestras con reemplazo del conjunto de datos original). Luego, se promedia las predicciones de estos modelos para obtener una predicción final. La principal ventaja de este método es que reduce la varianza al promediar múltiples modelos entrenados en conjuntos de datos ligeramente diferentes, lo que a menudo mejora el rendimiento predictivo en comparación con un solo modelo.

Es por ello que los modelos que mejor funcionan con Bagging son los árboles de decisión (concretamente, *fully grown*), que tienen una gran varianza y puede ser reducida mediante el promediado. De hecho, existe una variante de Bagging denominada *Random Forest*, de la que hablaremos luego. No obstante, también hemos probado Bagging con otros algoritmos, como KNN o Naive Bayes, aunque a priori sepamos que no van a dar mejores resultados.

Hemos decidido probar diferentes parámetros, tanto en el propio modelo individual como en Bagging. Los dos modelos de Bagging con árboles y con KNN son capaces de tener una precisión cercana al 100 % en training. Es interesante ver como KNN ha conseguido tener una precisión tan alta; no obstante, su rendimiento puede ser peor en el conjunto de test. Por otra parte, en Naive Bayes la precisión se mantiene constante incluso con 70 estimadores, y esto se debe a que Naive Bayes es un modelo con mucho bias y poca varianza, por lo que Bagging no puede mejorarlo a pesar de combinarlo muchas veces.

¿Qué ocurre si vemos el error en las muestras OOB?

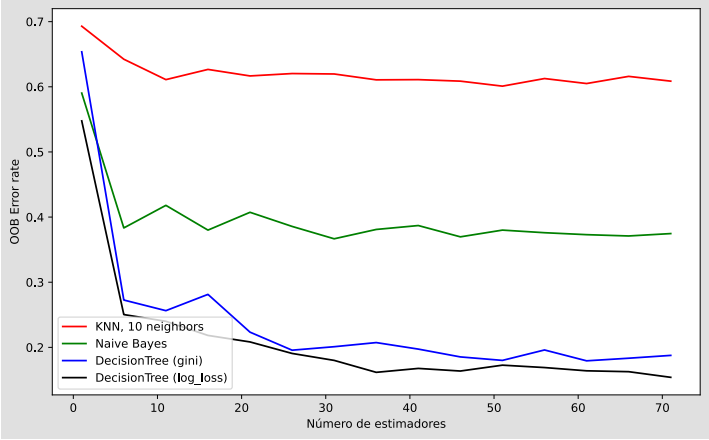


Figura 7: OOB error rate en Bagging

Aquí ya podemos observar que KNN no es capaz de generalizar a datos que no ha visto, por lo que su rendimiento en el conjunto de test será malo. Por otro lado, la precisión de Naive Bayes se mantiene parecida a la del conjunto de entrenamiento, como ya hemos mencionado. Y finalmente, los árboles son los que menor error tienen, especialmente el que utiliza el criterio *log_loss*.

### 2.2.2. RandomForest

RandomForest es un algoritmo basado en Bagging que utiliza como modelo el Árbol de Decisión. La diferencia clave con Bagging es que al elegir la característica que mejor divide los datos en cada nodo, no se comparan todas las $n$ características, sino un subconjunto $m$ de ellas, obtenidas de manera aleatoria. De esta manera reducimos la correlación entre los árboles, y obtenemos un ensemble con mayor capacidad de generalización.

En este caso hemos probado diferentes parámetros, especialmente el número de features $m$ que se escogían, número de estimadores, el criterio para dividir el nodo de un árbol y otros relacionados con el tamaño del árbol. Los árboles que mejor funcionan son los que no tienen limitación de tamaño, pues tienen una varianza muy alta, que es lo que mejor se ajusta al Bagging. Utilizar un decision stump aquí no tendría ningún sentido, pues el bias sería muy alto y no se podría corregir con los demás modelos.

En los datos de entrenamiento hemos observado que el modelo de Random Forest con árboles de profundidad 5 es demasiado pequeño como para aprender, y a pesar de que combinemos 100 estimadores se queda estancado en una precisión de 0.7 en el conjunto de entrenamiento. No obstante, los demás árboles llegan pronto al accuracy total, en tan sólo 15 estimadores. ¿Tiene sentido seguir entrenándolos? Veamos la precisión en muestras OOB.

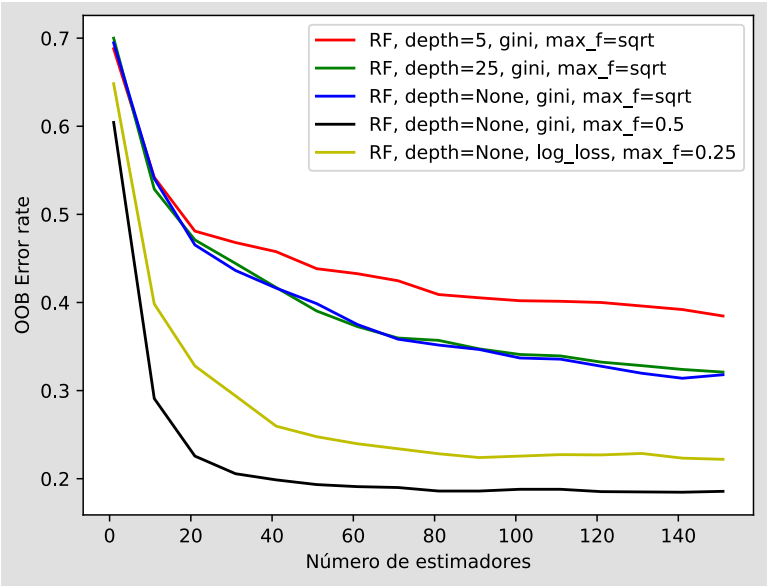


Figura 8: OOB error rate en Random Forest

Realmente sí, pues el error en las muestras OOB se va reduciendo el error hasta con 50 estimadores en todos los modelos. Concretamente, los modelos con parámetro $max\_features$ de 0.8 y 0.5 son los primeros en converger, pues valoran más variables a la hora de dividir los datos y pueden hacer mejores splits que los demás. Los otros 3 modelos siguen reduciendo su error y es posible que se acercaran a los demás, aunque con un número de estimadores más alto. Sin duda, el peor es el Random Forest que utiliza árboles poco profundos.

### 2.2.3. AdaBoost

AdaBoost es otro algoritmo de ensemble, ahora basado en la técnica de *boosting*. Los algoritmos de *boosting* combinan diferentes clasificadores débiles para formar uno fuerte. En cada iteración AdaBoost

actualiza los pesos de los datos, aumentando aquellos que no ha clasificado bien para que el siguiente clasificador débil les de prioridad. Cada clasificador también tiene su propio peso asociado, indicando qué tan bueno es.

Para entrenar AdaBoost, necesitamos clasificadores que sean al menos mejores que un predictor aleatorio. El método que utiliza por defecto es el árbol de decisión. Pero en este caso, no queremos un *fully grown tree*, sino un árbol más simple, como un *decision stump* o árboles con pocas divisiones (alto bias). Esto se debe a que necesitamos que el clasificador sea solamente mejor que la predicción random.

Por ello, hemos entrenado AdaBoost con árboles de diferentes tamaños. Además, veremos el rendimiento de AdaBoost discreto y Adaboost real, cuya diferencia radica en la forma en que se asignan los pesos y se combinan las predicciones de los clasificadores débiles. Después de probar diferentes modelos, nos hemos quedado con 5 de ellos, y hemos visto su evolución cuando aumentaba $m$:

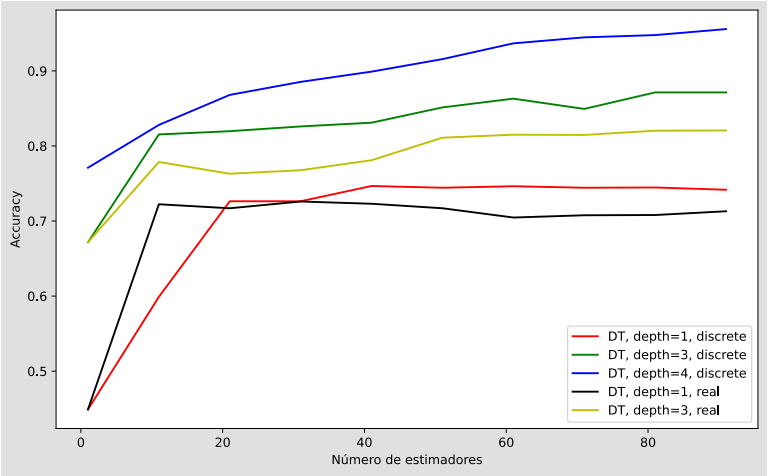


Figura 9: Resultados de varios modelos con AdaBoost en el conjunto de entrenamiento

En esta gráfica vemos como a partir de 20-30 estimadores todos los modelos se mantienen relativamente constantes, excepto el algoritmo discreto con profundidad 4, que parece que aún pueden aprender un poco más. No obstante, es posible que si sigue aprendiendo más acabe sufriendo overfitting, por ello hemos parado aquí. Es interesante ver que a medida que el $max\_depth$ es más alto, los árboles son capaces de ajustarse más al conjunto de entrenamiento. Pero esto no quiere decir que sean capaces de generalizar mejor, y después lo verificaremos.

No hemos mostrado modelos con más estimadores porque tardaba excesivo tiempo en mostrar la gráfica. No obstante, probaremos modelos más complejos en el conjunto de test para ver en qué punto empieza a haber overfitting y cuál es el número de estimadores óptimo.

### 2.2.4. GradientBoosting

GradientBoosting es otro algoritmo de *boosting*. En este caso, en cada iteración se ajusta un nuevo clasificador débil para predecir los errores del modelo combinado anterior, utilizando el algoritmo del descenso de gradiente sobre la función de pérdida. Al igual que AdaBoost, el clasificador débil suele ser un árbol de decisión de pequeño tamaño. No obstante, puede ser cualquier otro algoritmo de clasificación.

En este caso los parámetros más interesantes eran el tamaño del *subsample*, el $learning\_rate$, el $max\_features$ y el número de estimadores. Con respecto a $max\_features$, hemos decidido no dejarlo a None porque el tiempo de ejecución era muy alto. En los demás hemos probado diferentes opciones que han dado buenos resultados. Veamos cómo han evolucionado los distintos modelos probados a medida que aumentaba el número de estimadores.

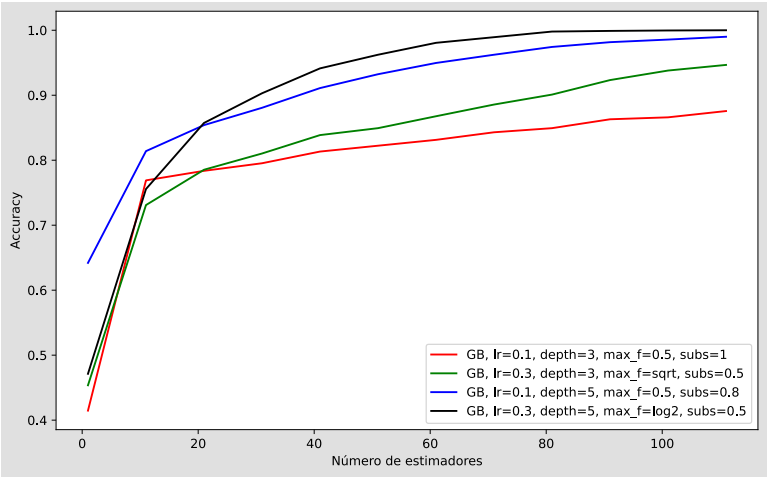


Figura 10: Resultados de varios modelos con GradientBoosting en el conjunto de entrenamiento

Podemos observar que los 4 modelos tienen una tendencia parecida, aunque finalmente al modelo rojo le cuesta más aprender, posiblemente por su bajo *learning_rate* y su bajo *max_depth*. El modelo que mejor se ajusta es el negro, que tiene una mayor profundidad en sus árboles y por ello, junto al azul, obtienen mejor precisión en el conjunto de entrenamiento.

# 3. Resultados

## 3.1. Mejores Resultados

| Classifier | Best Parameters | Train Accuracy | Test Accuracy |
|---|---|---|---|
| Naive Bayes | 6 Variables | 0.7023 | 0.7175 |
| Augmented Naives Bayes | scoring_type="BIC",prior_weight=20,... | 0.8473 | 0.625 |
| LDA | solver="lsqr",shrinkage='auto' | 0.442 | 0.405 |
| QDA | 3 Variables | 0.783 | 0.807 |
| Bagging (Decision Tree) | criterion="log_loss", depth=None, n_estimators=70 | 1 | 0.861 |
| RandomForest | criterion='gini', depth=None, n_estimators=150, max_features=0.5 | 1.0 | 0.824 |
| AdaBoost | algorithm='SAMME', depth=5, n_estimators=250 | 1 | 0.814 |
| GradientBoosting | lr=0.1, depth=5, max_features=0.5, subsample=1 | 0.993 | 0.808 |

Cuadro 2: Resultados

## 3.2. Imágenes complementarias

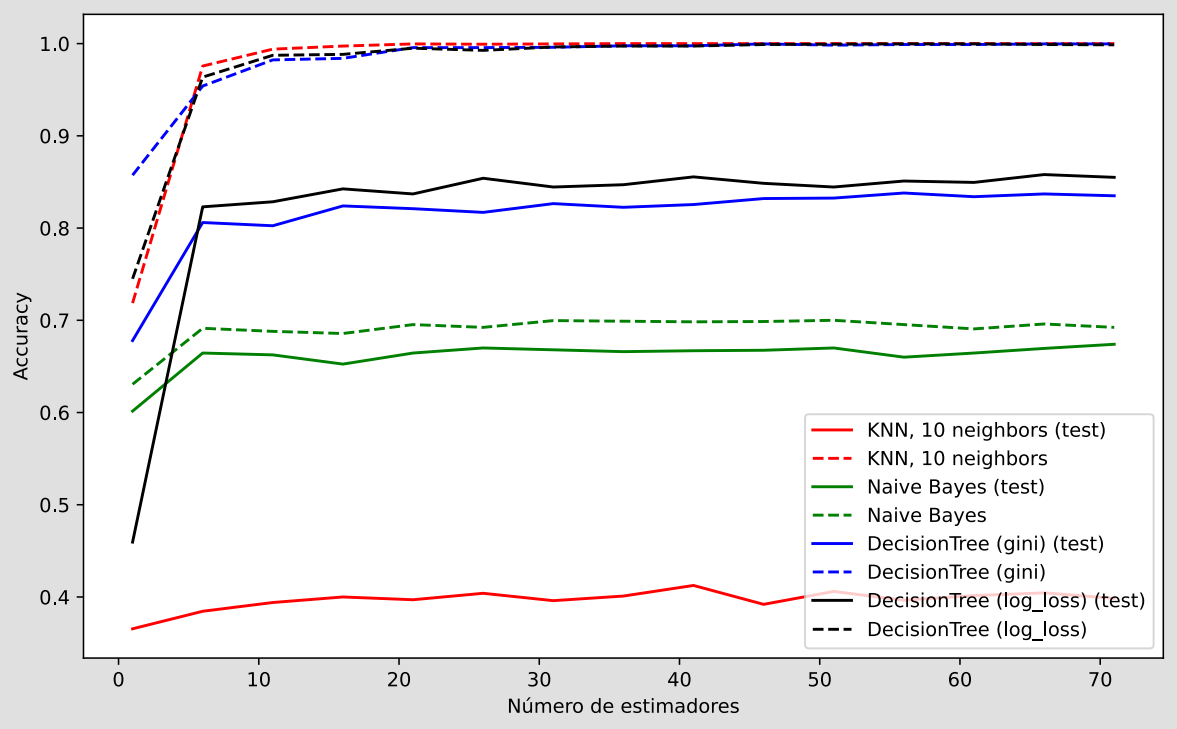


Figura 11: Accuracy en Bagging

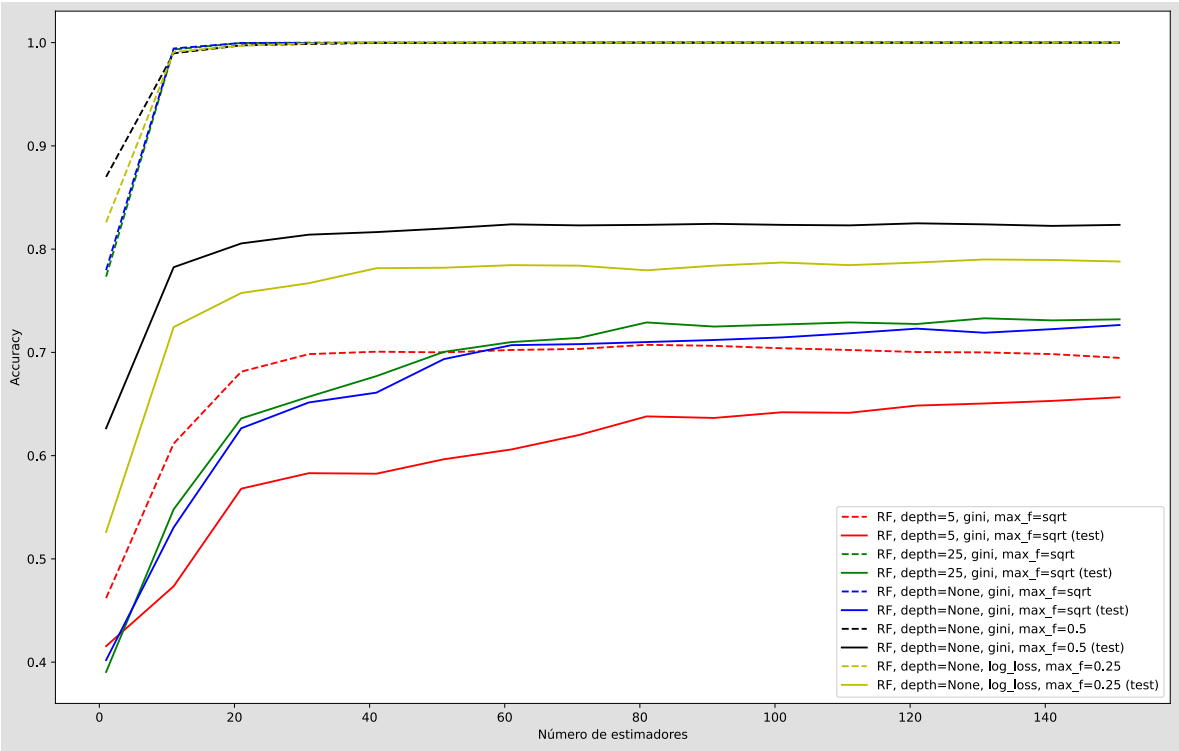


Figura 12: Accuracy en Random Forest

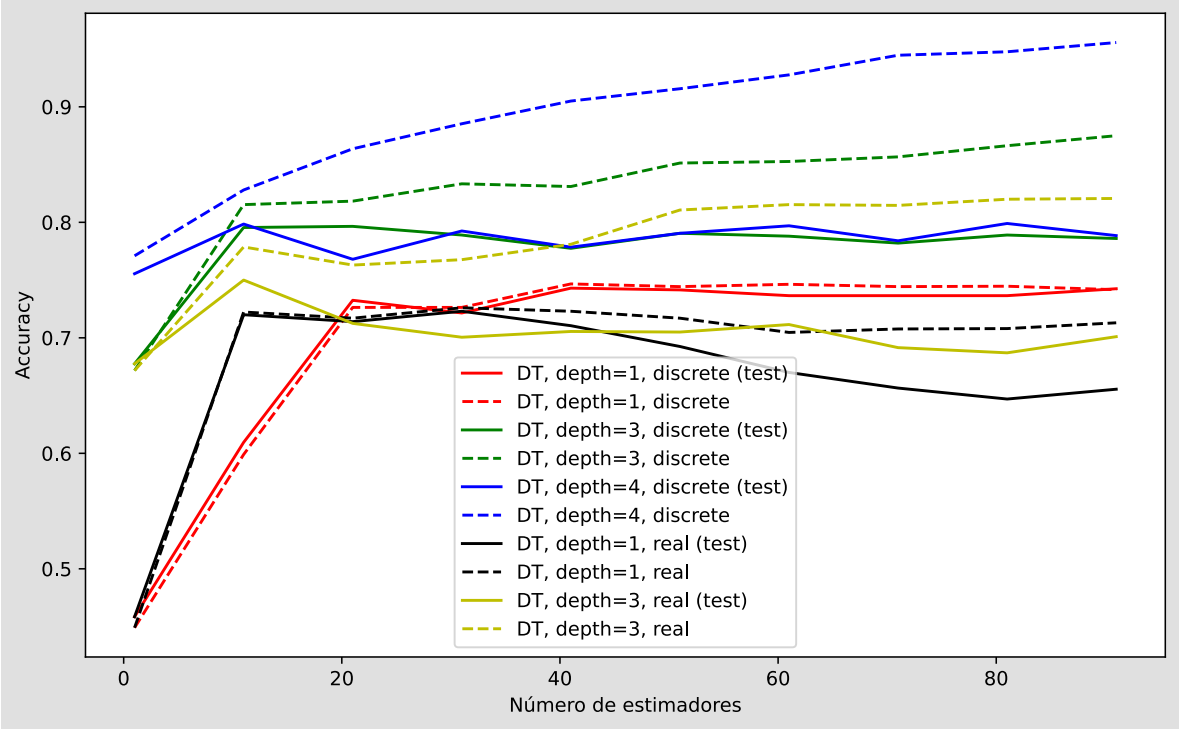


Figura 13: Accuracy en AdaBoost

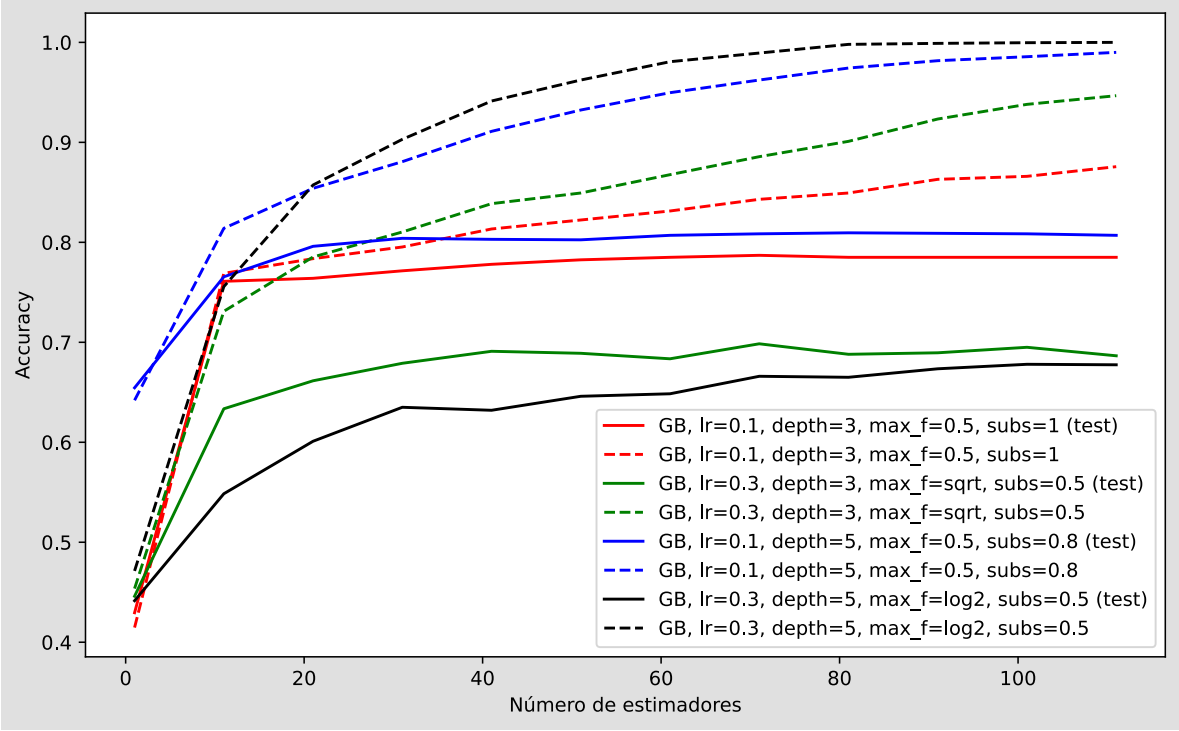


Figura 14: Accuracy en GradientBoosting

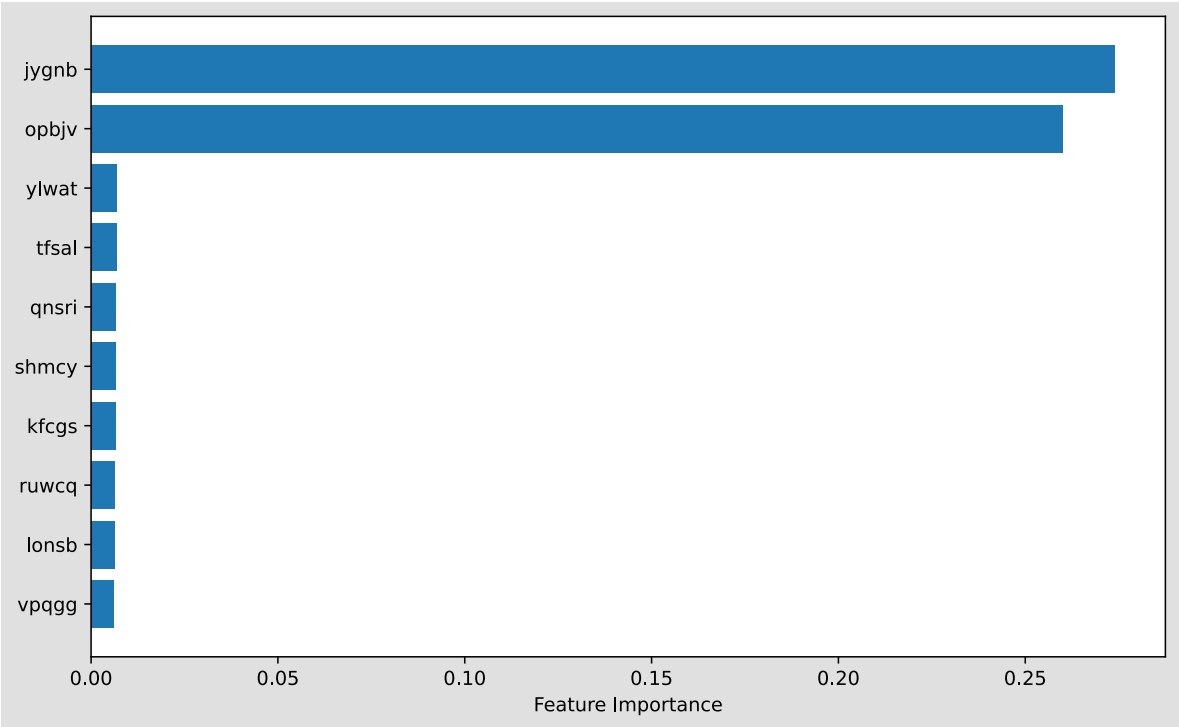


Figura 15: Feature importances de Random Forest

# 4. Discusión

Los mejores resultados en el conjunto de test han sido obtenidos por Bagging, pero vamos a ir comentando uno a uno cada modelo.

En el caso de Naive Bayes, el resultado en el conjunto de test ha sido de 0.7175 después de modificar el conjunto de entrenamiento para solo tener en cuenta 6 variables. No es un mal resultado para Naive Bayes, teniendo en cuenta que es un algoritmo que no suele dar los mejores resultados en ningún dataset. Hemos visto como eliminar las variables dependientes ha permitido reducir el error en el conjunto de entrenamiento, así como hacer selección de variables, dándonos cuenta de que realmente había muchas de ellas que no solo no aportaban gran información sino que empeoraban el modelo.

El Augmented Naive Bayes no ha dado los resultados esperados. Teníamos dos modelos, uno con muchas dependencias y otro con menos, y finalmente nos quedamos con el segundo por simplicidad y porque pensamos que muchas dependencias creadas por el primero podrían darse debido a los datos de entrenamiento, dando lugar a una peor generalización. Aún así, el modelo elegido dio unos resultados bastante pobres. Pensábamos que mejoraría el modelo Naive Bayes clásico gracias a relajar la asunción de independencia entre variables, aunque no fue así.

En cuanto a los modelos de Análisis Discriminante, podemos sacar conclusiones interesantes.

Los modelos de LDA creados dieron los peores resultados de todos, tanto con todas las variables como seleccionando variables. Creemos que esto se debe a las asunciones que hace LDA. En primer lugar, la asunción de normalidad, pues es posible que los datos de cada clase no se distribuyan como una función gaussiana en el espacio multidimensional. Y en segundo lugar, la asunción de que las clases comparten la misma matriz de covarianza. Esta asunción es muy fuerte e indica que los datos de cada clase se distribuyen con la misma forma, lo cual en muchas ocasiones no es cierto. Por ello, LDA ha obtenido una precisión tan baja, tanto en entrenamiento como en test.

Por otra parte, los modelos de QDA han sido una sorpresa. Tanto el modelo completo con todas las variables, como el modelo con las mejores variables seleccionadas han proporcionado una precisión muy alta en el conjunto de entrenamiento. Recordemos que QDA permite distintas matrices de covarianza, y es por ello que se puede ajustar tan bien a los datos de entrenamiento. Además, como vimos anteriormente con el segundo modelo, las matrices de covarianza eran diferentes para cada clase, lo que indica que los datos no se distribuyen de la misma manera en cada clase. También es interesante comparar el rendimiento de ambos modelos de QDA en el conjunto de test. Mientras que el primero de ellos, con todas las variables, ha proporcionado un resultado muy malo, el segundo ha conseguido generalizar obteniendo una precisión muy alta y cercana a los modelos de ensembles. Con tan solo 3 variables, el modelo se simplifica pero consigue un poder predictivo muy bueno, seguramente gracias a la mejor capacidad de generalizar y a la reducción del overfitting del primer modelo.

Pasando a los modelos de ensembles, en primer lugar hablaremos de Bagging. De los 4 modelos probados, no cabe duda de que los árboles de decisión son los que mejor rendimiento proporcionan. Es más, con una precisión superior al 85 % en el conjunto de test, son los modelos que mejor funcionan de entre todos los probados, específicamente el que trabaja con árboles utilizando el criterio 'log_loss'. Además, como vemos en la gráfica, su precisión se mantiene muy constante a partir de los 10 estimadores, aunque su OOB error rate continúa bajando incluso después de los 30 estimadores. En el caso de Bagging KNN, aunque el modelo se ajusta muy bien al conjunto de entrenamiento, su mal rendimiento en el conjunto de prueba sugiere sobreajuste. Esto puede deberse a un valor inapropiado del parámetro $k$. El parámetro $k$ en KNN es crucial para el balance entre bias y varianza: un $k$ muy pequeño puede causar alta varianza y Bagging podría lidiar mejor con ello (aunque podría dar lugar a sobreajuste), mientras que un $k$ muy grande puede causar alto sesgo y dar lugar a malos predictores individuales. Finalmente, con respecto a Naive Bayes, es interesante darnos cuenta de la poca varianza que hay: la diferencia entre la precisión en entrenamiento y en test es constante y muy pequeña, y esto se debe a la propia naturaleza de Naive Bayes, pues simplifica la complejidad del modelo asumiendo la independencia entre variables, dando lugar a predicciones robustas en datos que no se han visto.

Ahora pasamos a Random Forest, que también ha dado buenos resultados en el conjunto de test. Los 5 modelos probados, como ya se mencionó antes, diferían en la profundidad, criterio y número de características. Como ya preveíamos, el modelo con árboles de profundidad 5 ha dado resultados muy malos en el conjunto de test, pues su limitación le ha impedido aprender. En cuanto a los otros 4, la tendencia es parecida aunque a diferentes velocidades de convergencia. Como es lógico, los modelos que utilizan el parámetro $max\_features = sqrt$ necesitan más estimadores para alcanzar el mismo poder predictivo que los modelos que utilizan más características (0.5 y 0.25). Pero lo más seguro es que tarde o temprano alcanzarían la misma precisión en el conjunto de test. Además, hemos visualizado la importancia de las variables del mejor modelo de Random Forest conseguido, donde observamos que 2 de ellas sobresalen por encima de las demás: 'jygnb' y 'opbjv', con una importancia superior al 25 %. Las demás variables tienen una importancia parecida y muy pequeña, cercana al 1 %.

En cuanto a los modelos de Boosting, los resultados han sido ligeramente peores. Empezando por AdaBoost, los modelos que mejor rendimiento han dado en el conjunto de test han sido los que utilizan el algoritmo discreto ('SAMME'), concretamente los de profundidad más alta (3 y 4). Sabemos que los modelos de boosting funciona muy bien con clasificadores débiles, pero también es capaz de generar muy buenos modelos con clasificadores un poco más fuertes, como es este caso, en el que AdaBoost con el Decision Stump se ve superado por los otros dos. El algoritmo AdaBoost real ha proporcionado una precisión bastante baja en comparación, llegando apenas a un 70 %, aunque de nuevo siendo mejor el modelo con profundidad 3.

Finalmente, vamos con GradientBoosting. En este caso los 4 modelos se diferenciaban en la tasa de aprendizaje, la profundidad, el número de features utilizadas y el tamaño del subsample. Como se observa en la gráfica, los modelos con mayor número de features (0.5) son los que proporcionan una mejor precisión en el conjunto de test, cercana al 80 %, desde los 20 estimadores en adelante. Los otros dos modelos convergen más lentamente, ya que, a pesar de que su tasa de aprendizaje es mayor, el número de features que usan en cada clasificador es menor, y eso puede llevar a una precisión menor con el mismo número de estimadores que los otros modelos. No obstante, es posible que pudieran acercarse a ellos con un número más alto de estimadores.

# 5. Conclusión

Como hemos visto durante el desarrollo de la práctica, el modelo de Naive Bayes estándar tiene mejor resultado que el modelo Augmented Naive Bayes en este conjunto de datos. Esto sugiere que la simplicidad del Naive Bayes puede ser más adecuada para este conjunto en particular, posiblemente debido a que las suposiciones de independencia condicionales son suficientemente válidas, en especial cuando reducimos a 6 el número de variables.

Por otro lado, el modelo de Quadratic Discriminant Analysis (QDA) ha mostrado un mejor rendimiento en comparación con el Linear Discriminant Analysis (LDA). Esto indica que los datos dentro de cada clase tienen una distribución diferente, pues QDA modela con una matriz de covarianza cada clase de manera separada, captando así mejor las dificultades del conjunto de datos.

Además, la técnica de Bagging ha superado a la técnica de Boosting en este conjunto de datos. Esto puede sugerir que la reducción de la varianza y la robustez a los outliers y ruido que ofrece el Bagging son mejores para este caso, comparado con Boosting que se enfoca en corregir errores de modelos débiles.

Finalmente, los modelos ensemble han mostrado ser más estables en su desempeño en comparación con los modelos probabilísticos. La estabilidad de los ensembles, probablemente debido a la reducción de varianza y el efecto de promediar múltiples modelos, ha resultado en un rendimiento más consistente y confiable en este conjunto de datos.