UNIVERSIDAD DE GRANADA
E.T.S.I. INFORMÁTICA Y TELECOMUNICACIÓN

Aprendizaje Automático
Proyecto Final

Álvaro Fernández García
Salvador Corts Sánchez

3º Curso

Granada, curso 2017-2018

Índice

1. Problema a resolver

Vamos a trabajar sobre un conjunto de 41188 datos pertenecientes a la campaña de marketing de un banco. El objetivo es predecir si el cliente se subscribirá a un depósito a plazos a partir de los siguientes datos:

1. Edad del cliente.
2. Trabajo: *administrativo, cuello azul, emprendedor, empleado doméstico, directivo, retirado, autónomo, servicios, estudiante, técnico, desempleado, desconocido.*
3. Estado civil: *divorciado, casado, soltero, desconocido.*
4. Educación: *Básica 4 años, Básica 6 años, Básica 9 años, secundaria, analfabeto, curso profesional, universitario, desconocido.*
5. Morosidad: ¿debe algún crédito? *Si o No.*
6. Hipoteca: *Si o No.*
7. Préstamo Personal: *Si o No.*
8. Contactos: método de contacto con el cliente. *Teléfono Móvil o Fijo.*
9. Mes: en qué mes del año se le contactó por última vez. De *enero* a *diciembre.*
10. Día de la semana: día de la semana en el que se contacto con el cliente por última vez. De *lunes* a *martes.*
11. Duración: Cuanto ha durado la conversación con el cliente.
12. Campaña: cuantas veces se ha contactado con el cliente en esta campaña.
13. Pdays: días que pasaron entre que al cliente se le contacto para diferentes campañas.
14. Anteriores: Veces que el cliente fue contactado antes de esta campaña.
15. Resultado: Resultado de la anterior campaña. *fracaso, no-existe, éxito.*
16. Tasa de variación de empleo. Indicador trimestral.
17. Índice de precio del consumidor. Indicador mensual.
18. Índice de confianza en el consumidor. Indicador mensual.
19. Euribor a tres meses. Indicador diario.
20. Número de empleados. Indicador trimestral.

Como primer análisis debemos ver como de balanceada están las dos clases. Para ello, lo primero es contar el número de ejemplos de ambas clases:

- **Si**: 4640
- **No**: 36548

Como vemos estamos ante un claro caso de clases desbalanceadas, por lo que tendremos que tomar medidas al respecto teniendo en cuenta que para un banco es preferible un falso positivo que un falso negativo, es decir, el banco pierde más prediciendo que un cliente no va a contratar el depósito

cuando si lo haría que perdiendo el tiempo llamando a un cliente que finalmente dirá que no aunque hayamos predicho que si lo contratará.

También, cabe destacar que la característica ***Duración*** es muy informativa, pues un valor 0 (o cercano a el) nos indicaría un *No*. Sin embargo, esta es una característica que no nos sirve de nada ya que no se conoce a priori, si no a posteriori. La obviaremos.

Los autores del conjunto de datos aseguran que no faltan datos, sin embargo, hay valores *"Desconocido"* para las variables *Trabajo, Estado civil y Educación*. Tomaremos este valor como un posible valor más de la variable categórica.

2. Preprocesado de los datos

El preprocesado de los datos ha sido una de las partes más importantes de este proyecto. Nótese que la mayoría de las características que tienen los ejemplos del dataset son categóricas (concretamente un total de 10), y por tanto habrá que hacer un tratamiento apropiado de las mismas.

Por otra parte, las variables numéricas son todas de tipo entero, por lo que habrá que convertirlas a flotante.

2.1. Categorización

La categorización consiste en asignar una representación numérica a características categóricas para que los distintos modelos que utilicemos puedan trabajar con ellas. El proceso de categorización se divide en dos pasos fundamentales:

1. Transformar el rango de valores de la variable categórica en números enteros.

   A cada posible valor que toma dicha característica le asignamos un número entero que irá desde 0 hasta $n-1$, (donde n es el número de valores distintos que toma dicha característica). Por ejemplo, para la característica "Marital", que puede tomar como valores soltero, casado, divociado/viudo o desconocido, asignaremos el 0 a soltero, 1 a casado, 2 a divorciado/viudo y 3 para desconocido. Por tanto en todos los ejemplos donde aparezca "soltero", lo sustituiremos por un 0 e igual para el resto.

2. Aplicar "One Hot Encoder": el one hot enconder consiste en codificar las etiquetas enteras que hemos asignado en el apartado anterior utilizando un vector de unos y ceros.

   Por cada variable categórica, construiremos un vector de longitud n (donde n es el número de valores distintos que toma dicha característica). Para codificar un entero con ese vector, colocaremos un 1 en la posición del vector que coincida con el entero de esa variable y un 0 en el resto.

   Lo ilustraremos con un ejemplo: de nuevo tenemos nuestra característica "Marital", ahora con números enteros, y un ejemplo con el valor 0 (soltero). Con one hot encoder quedaría como:

   $$1 \quad 0 \quad 0 \quad 0$$

   Tiene tamaño 4 porque hay 4 posibles valores y ponemos un 1 en la posición 0 porque el 0 se asocia con soltero.

Una vez que hayamos hecho esto con todas las características categóricas, las concatenaremos todas junto con las numéricas y ya tendremos construida nuestra matriz X, la cual tiene un total de 62 características debido a todos los vectores añadidos por las categorizaciones.

También, con respecto al conjunto y, hemos asignado la etiqueta 1 al 'yes', y -1 al 'no'.

Nota 1: las variables categóricas binarias también se han codificado siguiendo esta técnica, ya que pueden considerarse como una categórica con dos etiquetas.

Nota 2: como ya hemos dicho antes, el categórico "unknown" se ha tomado también como un posible valor. Hemos considerado que es posible dentro del marco del problema, ya que el banco no siempre va a poder conocer todos estos datos del cliente al que desea preguntar.

2.2. Preparando Train y Test

Para realizar la división de los datos en train y test hemos considerado reservar un 20 % de los mismos para test y emplear el 80 % restante para train.

La partición se ha hecho de forma aleatoria pero manteniendo la proporción existente entre las etiquetas de yes (1) y no (-1). Esta decisión se debe fundamentalmente al gran desbalanceo que existe entre los ejemplos etiquetados como 1 y como -1, habiendo de estos últimos muchísimos más datos que de los del primero. Puede darse el caso de que todos los datos de train sean -1, así que hemos hecho esto para evitarlo.

2.3. Escalado y Normalización

Algunos modelos asumen que todas las características están centradas alrededor de 0 y tienen varianza en el mismo orden (normalmente una varianza de 1). Si una característica tiene una varianza que es de un órden de magnitud mayor que otras, podría dominar la función objetivo y hacer que el estimador no pueda aprender de otras características como se esperaba.

Una vez que hemos dividido los datos en train y test, crearemos una copia de los mismos y escalaremos las características flotantes (es de vital importacia que no escalemos las categóricas, ya que los 0 y 1 podrían quedar en valores inservibles, perdiéndose la codificación).

El motivo de mantener dos conjuntos (uno escalado y otro sin escalar) es porque utilizaremos modelos que requieren que los datos estén escalados (como la regresión lineal, SVM y Redes Neuronales) y otros que no (AdaBoost y Random Forest).

Por último, también es de vital importancia realizar el escalado de los datos de train y test por separado, ya que cuando se aplica el StandarScaler (que es el que hemos usado), utiliza la media de todos ellos. Si escaláramos antes de separar, los futuros datos de train estarían "observando" la media de los datos de test. Estaríamos mirando los datos de test de forma indirecta.

3. Métricas empleadas

Puesto que se trata de un problema de clasificación, en un principio pensamos que debíamos utilizar la métrica de Accuracy (la cual nos devuelve el porcentaje de predicciones correctas) así como la matriz de confusión (para observar con más detalle cuánto se equivoca y cuánto acierta en cada una de las clases).

El problema nos surgió precisamente cuando observábamos la matriz de confusión: las predicciones que estaba realizando eran muy pobres, es decir, de media los modelos estaban acertando un número razonable de ejemplos, pero también estaban errando demasiado en las predicciones afirmativas y como hemos mencionado anteriormente, el banco pierde más prediciendo que un cliente no va a contratar el depósito cuando sí lo haría (falsos negativos), que perdiendo el tiempo llamando a un cliente que finalmente dirá que no aunque hayamos predicho que sí lo contratará (falsos positivos).

El problema de esto reside en que tenemos clases muy desbalanceadas, y para esos casos, utilizar el accuracy tanto para estimar los hiperparámetros como elegir el mejor modelo puede tener pobres resultados. Claro que acierta mucho, la mayoría de los ejemplos son negativos, con que prediga una gran parte de los ejemplos como negativos va a obtener un buen accuracy, pero no está acertando casi ningún afirmativo. En definitiva queremos una métrica que nos de una aproximación más "realista" de como de bien está prediciendo ambas clases.

La solución ha sido utilizar como métrica el área bajo la curva ROC. La curva ROC suele tener el ratio de verdaderos positivos en el eje Y y el ratio de falsos positivos en el eje X. Esto significa que la esquina superior izquierda de la gráfica es el punto "ideal", un ratio de falsos positivos de cero y un ratio de verdaderos positivos de uno. Esto no es muy realista, pero sí significa que un área más grande debajo de la curva (AUC) denota un mejor comportamiento en el clasificador.

En definitiva, hemos utilizado dicha área para estimar los hiperparámetros y elegir el mejor modelo, no obstante no hemos eliminado el accuracy y la matriz de confusión, ya que cuanta más información tengamos, mejor podremos analizar el comportamiento del modelo.

4. Modelos, regularización y balanceo

Para este problema, hemos decidido aplicar todos aquellos clasificadores que conocemos: regresión logística, random forest, redes neuronales, SVM y también una ejecución sencilla de AdaBoost para ver como se comporta. Solo hemos dejado uno fuera, el perceptrón, debido a que hemos probado con support vector machine, el cuál es una versión mejorada del perceptrón.

Por otra parte, debido a que todos los problemas presentan ruido, y nunca estamos exentos de que se pueda producir sobreajuste, en todos los modelos aplicaremos algún tipo de regularización.

Por último, ya hemos mencionado que las clases están completamente desbalanceadas. Hay muchísimas instancias de *no*, pero muy pocas de *sí*. Para lidiar con este problema, a la hora de entrenar y validar los modelos le hemos dado más peso a la clase *sí*, que a la *no*. Concretamente, cada peso se calcula con la fórmula:

$$PesoClase_i = \frac{\text{Num Ejemplos}}{\text{Num Clases} \cdot \text{Num Instancias de la clase } i}$$

Donde el peso es inversamente proporcional a la frecuencia de esa clase en la muestra. De esta forma quedará balanceado en función del número de ejemplos de cada clase presentes en el dataset.

Solo hay una excepción en el procedimiento, AdaBoost, ya que la implementación del algoritmo en la librería usada no presenta ni regularización ni pesos para las clases. Realmente ya tenemos otros cuatro modelos con los que probar, no obstante ejecutaremos AdaBoost para ver como se comporta un modelo sin regularización y sin pesado de las clases sobre el dataset.

4.1. Regresión Logística

La regresión logística es un modelo lineal empleado para clasificación que nos devuelve la probabilidad que existe de que un determinado ejemplo pertenezca a cada una de las clases.

Este es uno de los motivos por los cuáles hemos pensado que podría resultar de utilidad aplicarlo para este problema: al banco puede interesarle saber realmente que probabilidad hay de que el cliente desee suscribirse. Si la respuesta es que *no*, pero la probabilidad de que sea que *no*, no es muy alta respecto a la de *sí*, a lo mejor sí que interesa comprobarlo.

Con respecto a la regularización, de las dos posibles (L1 o L2), hemos decidido quedarnos con L2. Es cierto que el problema tiene un gran número de características (62), pero la mayoría de esas características surgen a partir de la categorización realizada, y eliminar algunas de ellas puede no tener utilidad. Deberíamos haber eliminado las características irrelevantes antes de la categorización, pero solo teníamos 19, un número razonable. En definitiva regularizaremos con L2 para eliminar peso a aquellas características que no son muy relevantes, pero sin llegar a eliminarlas completamente.

Para estimar el hiperparámetro λ de regularización, hemos realizado validación cruzada con un total de 5 particiones, y probando con 10 valores elegidos aleatoriamente de una curva logarítmica que va desde 10^{-4} hasta 10^4. Después de la validación cruzada, se tiene que el número óptimo para el hiperparámetro de regularización es $\approx 0{,}046$.

Para terminar, después de entrenar con todos los datos de train, obtenemos los siguientes resultados: Accuracy = 82 %, área de la curva ROC = 0,74 y la siguiente matriz de confusión:

		Predicción	
		No	***Si***
Real	***No***	24932	4306
	Si	1364	2348

El accuracy y el área no son los mejores resultados pero tampoco son malos. No obstante tiene demasiados falsos negativos.

4.2. Support Vector Machine

Hemos decidido utilizar SVM porque es uno de los clasificadores que mejor comportamiento suele mostrar en una gran cantidad de problemas (a excepción del mundo de las señales, imágenes, audio...).

Al igual que en la regresión logística, aplicaremos regularización L2, y descartamos regularización L1 por las mismas razones: no interesa eliminar las variables añadidas por la categorización.

No obstante, con support vector machine tenemos un serio problema, y es que tenemos una cantidad de datos lo suficientemente grande como para que SVM sea terriblemente ineficiente.

En primer lugar debemos de determinar cuál es el λ apropiado para regularización, también podemos probar con dos kernels: el bayesiano y el polinomial. Además, para estos kernels, hay que determinar otro parámetro libre: γ. Para determinar todo esto, hay que realizar validación cruzada y así encontrar la mejor combinación de hiperparámetros. Esto, unido a la gran cantidad de datos, hace que sea impracticable.

Para solucionarlo hemos tomado las siguientes medidas:

- Hemos probado únicamente con el kernel bayesiano, ya que el polinomial consumía un tiempo bastante más elevado que el primero.
- Para los λ hemos usado los mismos que en regresión logística (10 valores aleatorios de una curva logarítmica entre 10^{-4} y 10^4) y para γ, 5 valores aleatorios entre 10^{-2} y 10 (ya que solo se permite una precisión de 2 decimales o enteros).
- Hemos realizado validación cruzada dividiendo en 5 partes pero sólo sobre un subconjunto de la muestra, concretamente para 5000 datos seleccionados aleatoriamente sin repetición. Con esos datos probamos todas las combinaciones posibles, para quedarnos con las 3 mejores combinaciones.
- Por último, probamos las tres mejores combinaciones sobre todos los datos con validación cruzada y 5 particiones, para devolver la mejor de todas.

Tras realizar este proceso, tenemos que los mejores hiperparámetros para el kernel bayesiano son: $\lambda \approx 0,36$ y $\gamma = 0{,}01$. Con estos valores se obtienen los siguientes resultados: Acurracy $\approx 83\,\%$, área bajo la curva ROC $\approx 0{,}74$ y matriz de confusión:

		Predicción	
		No	***Si***
Real	***No***	25192	4046
	Si	1405	2307

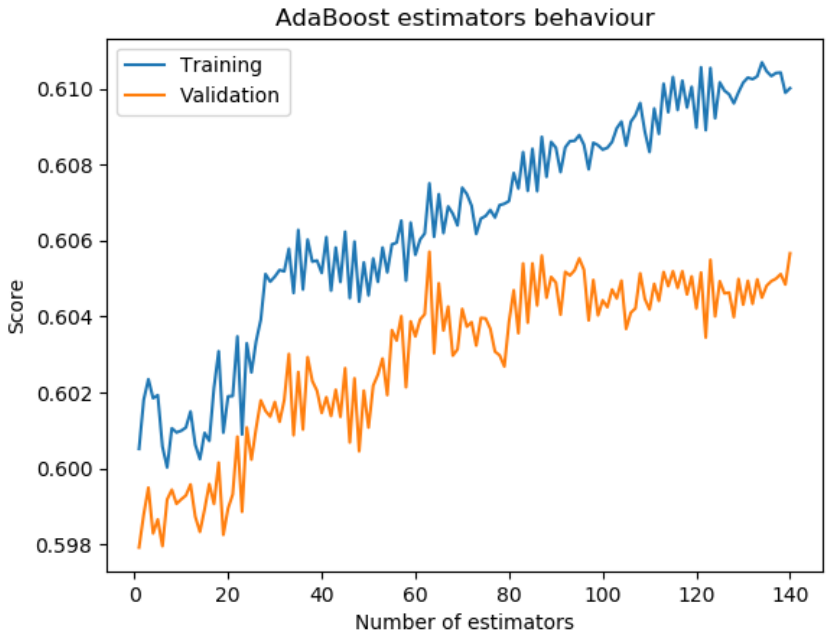
No presenta mucha diferencia frente a la regresión logística.

4.3. AdaBoost

Otro de los modelos que aproximaremos será AdaBoost. Se trata de un metaestimador que comienza ajustando un clasificador en el conjunto de datos y luego ajusta copias adicionales del clasificador sobre el mismo conjunto de datos, pero donde los pesos de las instancias clasificadas incorrectamente se ajustan de modo que los clasificadores posteriores se centren más en los casos difíciles.

El principal hiperparámetro de AdaBoost a ajustar es el número de estimadores que utilizará. No basta con elegir un número aleatorio grande, hay que elegirlo inteligentemente, evitando el sobreajuste y minimizando el error. Para ello, hemos diseñado una función que calcula para un modelo basado en varios estimadores (como AdaBoost y Random Forest) el número óptimo de clasificadores.

Irá incrementando el número de estimadores y, mediante validación cruzada, irá viendo como evoluciona el error de entrenamiento y validación. En el momento que el error de entrenamiento disminuya y el de validación aumente pararemos pues comienza el sobreajuste. Como error tomaremos el area bajo la curva ROC.



Como vemos del anterior gráfico no podemos obtener el número de estimadores a partir del cual comienza el sobreajuste, ya que ambos errores crecen o decrecen simultáneamente en función del número de estimadores. Sin embargo, si podemos sacar algo en claro, para un número de estimadores entre 0 y 150 no se produce overfitting por lo que podemos escoger el que mejor error en validación nos de. Utilizaremos el método de *Scikit-Learn* ***RandomizedSearchCV*** el cual probará valores aleatorios para el número de estimadores dentro de este rango y se quedará con el mejor.

Obtenemos que el número óptimo de estimadores en ese rango es 130. Con esto, obtenemos un modelo con una precisión del 90 %, un área bajo la curva ROC de $\approx 0{,}61$ y la siguiente matriz de confusión sobre los datos de entrenamiento:

		Predicción	
		No	***Si***
Real	***No***	28850	388
	Si	2868	844

Podemos ver que AdaBoost no ha tenido mucho éxito. Aunque su Accuracy es bueno, la cantidad de falsos negativos es demasiado elevada. En definitiva, el área bajo la curva ROC no es muy alta, lo que se traduce en un peor comportamiento del clasificador con respecto a los probados anteriormente.

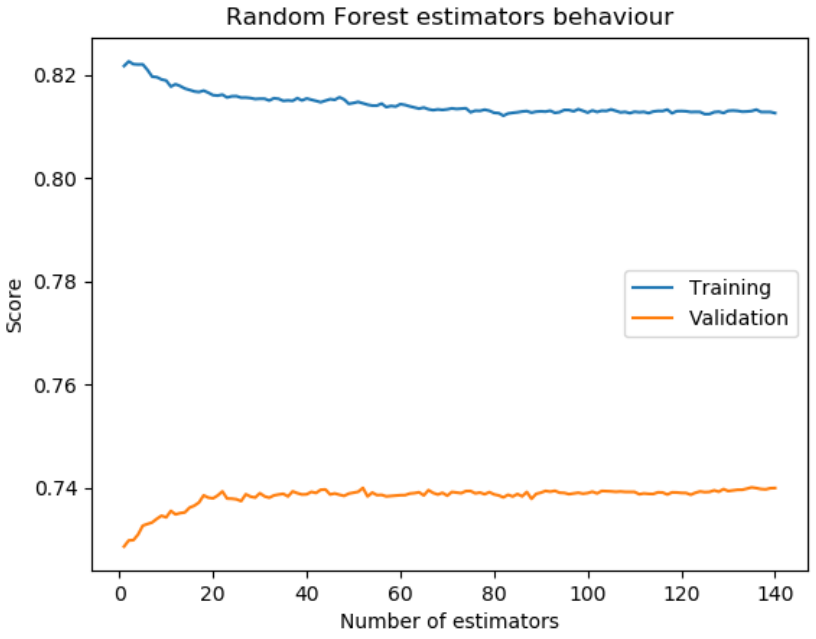
4.4. Random Forest

Random Forest se trata de un metaestimador que ajusta un número determinado de árboles de decisión a varias submuestras. Una característica muy interesante de Random Forest y que usaremos para entrenar nuestro modelo es el concepto de ***Bootstrap*** que nos permitirá mejorar la estabilidad y precisión de nuestro modelo, así como reducir la varianza.

En cuanto a los hiperparámetros de Random forest, debemos especificarle que use la entropía como

criterio para medir la calidad de una separación, y que seleccione un total de $\sqrt{p}$ características (donde p es el número total de ellas). Por otro lado, al igual que con AdaBoost, otro hiperparámetro muy importante es el número de estimadores (árboles de decisión) que usará nuestro modelo. Utilizaremos la misma función que con AdaBoost para encontrar el número de arboles a partir del cual el modelo empieza a sobreajustarse y por lo tanto el error de validación aumenta mientras que el de entrenamiento disminuye. De nuevo utilizaremos el área bajo la curva ROC para medir el error.

A continuación podemos ver como evoluciona el error de validación y el de entrenamiento en función del número de árboles:



De nuevo podemos ver que no nos es de mucha ayuda de cara a obtener el número de árboles. Sin embargo nuevamente podemos extraer la conclusión que en este rango de árboles (1, 150), no se produce sobreajuste, por lo tanto dejaremos a *Scikit-Learn* decidir el número de estimadores mediante la función ***RandomizedSearchCV***.

Hemos tenido problemas de sobreajuste con este modelo y para evitarlo decidimos forzar una poda mayor en el árbol. Mediante prueba y error hemos limitado la profundidad máxima de los árboles y el número de hojas máximo. Finalmente hemos obtenido un buen resultado con una profundidad máxima de 1000 y 1000 nodos hoja.

Con un modelo de 130 árboles con poda y ponderación de clases ajustado sobre los datos de entrenamiento obtenemos una precisión del 89 %, un área bajo la curva ROC de 0.79 y la siguiente matriz de confusión:

		Predicción	
		No	***Si***
Real	***No***	26870	2368
	Si	1208	2504

Como vemos estamos ante un modelo relativamente bien ajustado. No solo obtenemos una buena

precisión, además conseguimos un buen resultado ROC.

Como comentario final sobre Random Forest, en problemas relacionados con la banca o la medicina, no se suelen usar cajas negras ya que suele haber algo muy delicado en juego (la vida de una persona o una inversión). Sin embargo en este caso, una caja negra como Random Forest es totalmente aceptable ya que el banco no se está jugando nada, simplemente quiere saber si un cliente contratará un depósito a plazos.

4.5. Redes Neuronales

Dada la complejidad del problema, quisimos probar como de bien funcionaba un Perceptrón Multicapa. Debido a la dificultad a la hora de encontrar una topología adecuada, hemos optado por escoger 4 combinaciones de número de capas ocultas y neuronas en cada una.

- Una capa intermedia con 30 neuronas.
- Dos capas intermedias con 30 neuronas.
- Tres capas intermedias con 30 neuronas.
- Tres capas; la primera con 30, la segunda con 20 y la última con 10 neuronas.

Mediante validación cruzada, maximizaremos el valor del área bajo la curva ROC combinando dichas topologías con los activadores *sigmoide, tangente hiperbólica y relu*, así como con un valor λ para regularización L2 dentro de la curva logarítmica que va desde 10^{-4} hasta 10^4.

La mejor combinación encontrada ha sido:

- **Topología**: Una capa intermedia con 30 neuronas.
- **Activador**: Tangente hiperbólica.
- λ **para regularización L2**: 0.005994842503189409

Con esto hemos obtenido una precisión del 90 %, un área bajo la curva ROC del $\approx$ 0,61 y la siguiente matriz de confusión:

		Predicción	
		No	***Si***
Real	***No***	28902	336
	Si	2874	838

Al igual que con AdaBoost, aunque la precisión es buena, no somos capaces de predecir igual de bien ambas clases. De hecho, erramos significativamente más en la clase *Sí* que es la más importante para el Banco.

5. Modelo Final y conclusiones

A continuación se muestra una comparativa de los resultados obtenidos para cada modelo ajustado sobre los datos de entrenamiento:

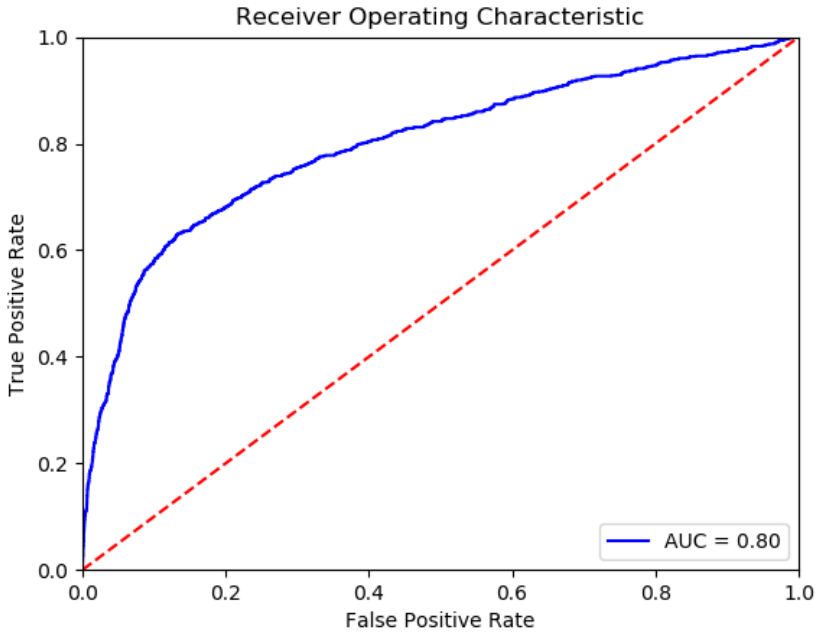
Modelo	**Accuracy**	**Área Roc**
Regresión Logística	82 %	0,74
SVM	83 %	0,74
AdaBoost	90 %	0,61
Random Forest	89 %	0,79
Redes Neuronales	90 %	0,61

Como ya hemos dicho en el apartado de Métricas, para problemas con conjuntos de datos desbalanceados no es apropiado dejarnos guiar por el Accuracy, por este motivo, para seleccionar el mejor modelo de todos, nos quedaremos con aquel que presente mayor área bajo la curva ROC, ya que denota un mejor comportamiento. Ese modelo es **Random Forest**.

Veamos que tal se comporta con datos nunca antes vistos por el modelo. Tras realizar el test, obtenemos una precisión del 87 %, un área bajo la curva ROC de 0.80 y la siguiente matriz de confusión:

		Predicción	
		No	***Si***
Real	***No***	6637	673
	Si	401	527

También se muestra la gráfica que representa la cruva ROC:



Como podemos comprobar, estamos ante un buen ajuste (no es perfecto, pero es bastante bueno), la curva azul esta por encima de la línea roja. Dicha linea roja, denominada línea de no-discriminación, es la correspondiente a un modelo que predice clases aleatoriamente. Cualquier punto situado por encima de esta linea de no-discriminación denota un modelo mejor que uno aleatorio.

La cuestión es, ¿un área bajo la curva ROC del 0.80 es lo suficientemente buena para nuestro problema? Teniendo en cuenta que el dataset proporcionado está terriblemente desbalanceado, y

que empíricamente la mayoría de los modelos ajustados (a excepción de las Redes Neuronales y AdaBoost que no han tenido un comportamiento muy bondadoso) dan resultados más o menos similares (entre 0.74-0.79 de curva ROC y 80-89 % de accuracy), podemos deducir que es poco probable obtener un modelo de mejor calidad para los datos proporcionados. Además, no es un problema que tenga fuertes restricciones sobre los errores (como por ejemplo, un problema de predicción médica). Como ya dijimos en el primer punto, es preferible un falso positivo que un falso negativo, es decir, el banco pierde más prediciendo que un cliente no va a contratar el depósito cuando sí lo haría que perdiendo el tiempo llamando a un cliente que finalmente dirá que no aunque hayamos predicho que sí lo contratará, y hemos centrado nuestros esfuerzos en intentar que el clasificador prediga de forma apropiada los casos positivos, por lo que podemos concluir que sí, se trata de un buen ajuste.