Aprendizaje Automático. Proyecto final

Manuel Herrera Ojea
53583380G

Ismael Marín Molina
50627728J

Contents

1 Descripción del problema **2**

2 Conjunto de datos utilizado **2**

3 Modelos empleados **3**
3.1 Random Forest 3
3.2 Motivación para RF 3
3.3 Support Vector Machine 3
3.4 Motivación para SVM 4
3.5 Combinación de modelos 4

4 Desarrollo de la clasificación **4**
4.1 Tratamiento de los datos 4
4.1.1 Valores Perdidos 5
4.1.2 Desbalanceo del modelo 6
4.1.3 Valores Categóricos 8
4.2 Toma de hiperparámetros 9

5 Resultados **10**

1 Descripción del problema

Nuestro dataset se trataba de los datos recopilados de diferentes clientes del banco, para conocer su interes en un nuevo tipo de depósito lanzado.

Para ello nos hemos usado la base de datos Bank Marketing Data Set (BMDS), alojado en el repositorio de aprendizaje automático UCI en `https://archive.ics.uci.edu/ml/datasets/bank+marketing`.

2 Conjunto de datos utilizado

El conjunto BMDS consta de 41188 instancias, cada una correspondiente a los datos obtenidos por vía telefónica de los clientes de una institución bancaria portuguesa. Durante la extracción de datos no fue posible obtener todo de todos los clientes, por lo que el conjunto de aprendizaje presenta valores nulos en ciertas características, representados por la cadena "`unknown`", este dataset también trae datos de otros años, pero nosotros nos hemos centrado en el año más actual, 2014.

Los datos tomados por la entidad bancaria muestran 20 características de cada uno de sus clientes, teniendo en cuenta entre ellas los posibles valores nulos recién mencionados: (1) edad, (2) ocupación, (3) estado civil, (4) nivel de estudios, (5) crédito por defecto, (6) petición de un crédito inmobiliario, (7) petición de un crédito personal, (8) método de comunicación con el banco, (9) mes, (10) día de la semana y (11) duración de la última comunicación (si la duración de la conversación es 0, es decir no cogió el telefono, entonces se predice como "no"), (12) cantidad de contactos realizados con el banco, (13) días transcurridos desde el último contacto, (14) veces contactado antes de la campaña para la suscripción, (15) resultado de la campaña de suscripción anterior, (16) tasa de variación de empleo, (17) índice de precios de consumo, (18) índice de confianza de consumo, (19) euribor a tres meses, y (20) cantidad de empleados. La característica a predecir es (21) Interés en el producto , tomando los valores "si" o "no".

De estas características, las número (2), (3), (4), (5), (6), (7), (8), (9), (10) y (15) son categóricas, presentándose en formato cadena de texto, mientras que el resto son valores numéricos.

3 Modelos empleados

3.1 Random Forest

RF es un modelo de aprendizaje automático que usa un conjunto de árboles de decisión muy simples que van dado una predicción poco realista pero combinando las predicciones de todos los arboles que lo conforman da una predicción bastante buena, para combinar los datos en el caso concreto de los arboles de la `sklearn` para problemas de clasificación, usa el "valor más votado".

Esto arroja una predicción más fiable que la de un solo árbol de decisión, pues disminuye drásticamente la probabilidad de que caigan en un sobreajuste durante el aprendizaje, problema que suele surgir al utilizar árboles de decisión debido a la alta varianza que estos tienen.

3.2 Motivación para RF

La existencia de tantas características categóricas en nuestro conjunto de datos ha sido un gran motivador de la elección del modelo de aprendizaje RF, por ser los árboles de decisión, maquinaria interna de la predicción realizada por un RF, especialmente potentes para realizar predicciones tomando valores categóricos, pues la naturaleza discreta de sus ramificaciones se acopla muy bien a atributos cuyo dominio no sea continuo.

Otro factor que ha influido en nuestra decisión de utilizar RF es que a diferencia del resto de modelos, RF (más concretamente Decisión Tree) nos da aquellas características que más efecto han tenido a la hora de clasificar el problema, lo cual analizando la naturaleza comercial de nuestro problema creemos es algo muy significativo.

El último motivo es que los RF nos darán una aproximación muy buena del modelo con una capacidad de computo más reducida.

3.3 Support Vector Machine

El modelo SVM es un modelo el cual se basa en encontrar la mejor separación posible entre las distintas clases, para ello se apoya en la busqueda de aquellas instancias que definan el "pasillo" existente entre las distintas clases.

Para la creación de esta separación, conociendo que la mayoría de datos reales no son separables de forma lineal, usa una transformación de los valores con el uso de una función la cual elegimos nosotros mismos, esta función es llamada kernel.

3.4 Motivación para SVM

Hemos pensado en el uso del SVM al tratarse de un problema de clasificación binaria, con la precensia de muchos características reales, podría funcionar correctamente. También decir que este modelo era recomendado en la descripción del dataset.

3.5 Combinación de modelos

Como último modelo hemos provado una combinación de ambos modelos antes mencionados, para ello hemos actuado de la siguiente forma y con el con siguiente razonamiento.

Hemos usado RF para iniciar una primera aproximación del error en la muestra de training, en aquellas instancias en las que RF haya fallado, quiere decir que la mayoría de los arboles usados en este han mal clasificado la instancia, entonces hemos decidido quitarla al tomandola como una instancia ruidosa y reduciendo la dimensión del conjunto de entrenamiento que será pasado a SVM.

Haciendo uso del RF que ya tenemos entrenado y conociendo la capacidad explicativa de este modelo, hemos obtenido las características más relevantes para la clasificación y estas, y solo estas han sido las utilizadas para entrenar el modelo SVM.

Con esta combinación de los modelos hemos podido explotar más las capacidades del SVM y crear un modelo más acertado.

Table 1: Distribucion valores perdidos

Ocupación	Estado C.	Nivel Estudios	Crédito por defecto	PCI	PCP
0.801%	0.194%	4.202%	20.872%	2.403%	2.403%

4 Desarrollo de la clasificación

4.1 Tratamiento de los datos

Ha sido necesario un preprocesamiento de los datos facilitados por BMDS para su correcto tratamiento por las funciones implementadas en el framework scikit-learn. Primero ha sido necesario arreglar los valores perdidos. Después hemos corregido el desbalanceo entre instancias de cada clase, tras ello una codificación de las características categóricas y finalmente una estandarización de las características numéricas. Esto ha sido realizado tanto en la partición de training como en la de test, las cuales han sido divididas en un 70% del tamaño para Training y Validación y un 30% para el test.

4.1.1 Valores Perdidos

Cuando se genero la base de datos BMDS, como hemos mencionado anteriormente, no fue posible obtener todos los datos requeridos de todos los clientes, en ocasiones ni tras haber realizado el banco varios contactos con un mismo cliente. Es por ello que hay valores desconocidos, que están registrados como la cadena de texto `'unknown'`. Esto tiene una semántica implícita muy clara, pero los modelos de `scikit-learn` no son capaces de diferenciarla del resto de valores posibles, por eso hemos tenido que tratarlas a mano.

Inicialmente hemos hecho un conteo del número de valores perdidos en cada característica, para comprobar como estabán distribuidas en nuestro data set, el resultado fue el siguiente:

Como podemos comprobar en la tabla 1 la mayoría de estos se concentraban en la columna del Crédito por defecto. Con una menor representación tenemos las columnas de prestamó de credito inmobiliario (PCI) y prestamó de credito personal (PCP), junto con el nivel de Estudios, con una menor cantidad tenemos la ocupación y el estado civil.

Para realizar la imputación de estos valores perdidos de la manera más eficaz posible sin realizar un excesivo computo hemos dividido los valores

perdidos entre aquellos que tenián un más de un 1% de los valores de su clase perdidos y los que tenían menos.

Aquellos con menos de esta cantidad hemos decidido imputarlo usando el valor más frecuente, el cual nos parece un mejor estimador de la realidad de la población que la media o mediana.

El resto hemos decidido usar una táctica más sofisticada imputandolos haciendo uso de un 3-NN, para predecir cual será su valor. Para ello hemos separado los datos según si tenian o no el valor, hemos entrenado el modelo con los datos que no tenian valores perdidos y hemos cambiado los "unknown" por los valores predecidos por el modelo.

Solucionando los 12718 valores perdidos que teniamos

4.1.2 Desbalanceo del modelo

En BMDS, de las 41188 instancias facilitadas, 36548 pertenecen a la clase negativa y solo 4640 a la clase positiva, cuando lo ideal es que hubiese habido una representación aproximadamente equitativa de ambas clases.
Tras el tratamiento de los valores perdidos hemos comprobado en el conjunto de training un fuerte desbalanceo entre ambas clases, este desbalanceo provoca que el modelo no pueda predecir bien la clase minoritaría, lo cual hace que aunque la calidad del modelo parezca acertada, la realidad es que en la minoritaría el acierto es mínimo. En nuestro caso el modelo acertaba el 95% de las pertenecientes a la clase 0 pero solo el 50% de aquellas que estaban en la clase 1.

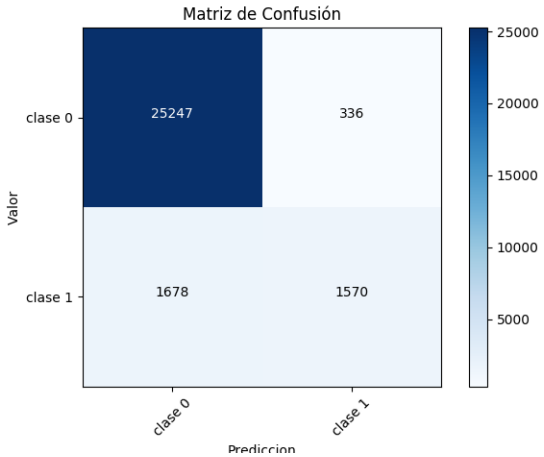

Figure 1: Matriz de Confusión Desbalanceada

Para solucionar esto hemos usado el algoritmo SMOTE, siglas de Synthetic Minority Over-sampling Technique. Es una técnica utilizada en conjuntos donde las representaciones de cada clase no están equilibradas.

Los desarrolladores de SMOTE han estudiado este problema, y su herramienta, en lugar de únicamente reducir la cantidad de instancias de la clase mayoritaria, en nuestro casos los `'no'`, hasta alcanzar una representación equitativa, ofrecen un método con el que aumentar la cantidad de instancias de la clase minoritaria. Esto es importante, pues tratar únicamente la poda de instancias negativas habría llevado a aumentar significativamente el comportamiento de nuestro predictor para favorecer los casos positivos más de lo deseado. Para aumentar la cantidad de instancias de la clase minoritaria se han servido de métodos sintéticos de generación de instancias. Según los estudios que han realizado, tanto con Ripper como con clasificadores bayesianos ingenuos, los resultados han sido mucho más fructíferos al aplicar a conjuntos no equilibrados este tipo de poda y generación que al aplicarle únicamente podas.

El algoritmo SMOTE funciona creando ejemplos sintéticos de la clase minoritaría para equilibrar ambas clases, para ello usa un KNN interno el cual toma los KNN vecinos más cercanos y genera un ejemplo usando alguno de ellos. El pseudocódigo de este algoritmo es el siguiente, sacado del paper presentado por Nitesh V. Chawla y Kevin W Bowyer en 2002:

La implementación de este algoritmo hemos usado la librería `imbalanced learn`, con los parametrós de $k = 3$ y usar la función de creación `borderline 1`. La cual es la función descrita en el pseudocódigo.

Algoritmo 1 SMOTE

Entrada: T = Poblacion de la menor clase ,N = numero de ejemplos sinteticos, k = numero de elementos del kNN.

Salida: Array con los ejemplos sinteticos.

1: **para todo** $i \in T$ **hacer**
2: Computamos los k vecinos más cercanos y guardamos el indice en $nnarray$.
3: Populate(N, i, $nnarray$)
4: **fin para**
5: **mientras** $N \neq 0$ **hacer**
6: Elegimos un vecino aleatorio de la instancia i
7: **para todo** Atributo en el ejemplo i **hacer**
8: Compute: $dif = Sample[nnarray][nn][attr] - Sample[i][attr]$
9: Compute: gap = random number entre 0 y 1
10: Compute: $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
11: **fin para**
12: $newindex++$
13: $N--$
14: **fin mientras**
15: **devolver** Population Synthetic

Las clases han sido equilibradas hasta igualar ambas dejando un total de 48818 instancias en el conjunto de training. Tras volver a realizar la matriz de confución del conjunto de entrenamiento estos son los nuevos resultados obtenidos.

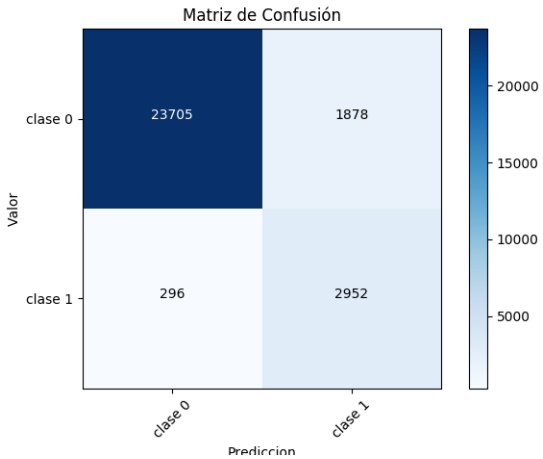

Figure 2: Matriz de Confusión Balanceada

4.1.3 Valores Categóricos

Otra convención que utiliza scikit-learn es utilizar valores numéricos enteros para la codificación de variables categóricas. Los valores de catacterísticas catagóricas de BMDS están codificadas todas como cadenas de texto, por lo que ha sido necesaria una conversión. Para ello hemos utilizado el codificador one-hot, implementado también dentro de scikit-learn.

La forma de proceder del codificador one-hot es la siguiente. En lugar de simplemente asignar valores enteros a los posibles valores del dominio de una característica categórica, crea una nueva característica por cada valor que pueda tomar una categoría. Esto aumenta considerablemente la dimensionalidad del problema, pero arroja unos resultados mucho más satisfactorios.

El principal problema que tiene codificar los valores de una característica categórica como valores enteros es que se establece inherentemente una relación de orden entre ellos, y el modelo puede aprender que un mayor o un menor valor de dicha característica es el preferible, o buscar un valor concreto y

beneficiar colateralmente a los valores colindantes. Esto es problemático porque el orden en el que se hayan codificado los valores de la característica categórica influye directamente en la respuesta que dará el predictor, cuando una permitación de estos valores no debería influir de forma apriorística al aprendizaje.

La solución que ofrece la codificación one-hot, crear una nueva característica por cada valor posible, hace que no existan relaciones implícitas entre estos valores antes de que comience el aprendizaje. Cada dato de entrada tendrá un valor 1 en la característica correspondiente al valor que tenía antes, y 0 en el resto de características que se hayan creado que equivalgan a los demás posibles valores que podía haber tomado este dato en esta característica. Así, de las nuevas características que se crean por cada característica categórica originaria, solo en una trendrá el dato transformado el valor 1, y 0 en todos los demás.

Este proceso se conoce también como binarización de una variable categórica, donde se convierte una variable con n posibles valores en algún dominio en n variables binarias.

4.2 Toma de hiperparámetros

Para determinar los mejores hiperparámetros hemos usado el metalgoritmo de la `metrics.gridSearch`, el determinamos los parametros a probar con los distintos modelos en el conjunto de entrenamiento y nos devolverá aquella combinación de parámetros que mejores resultados nos den. En el caso del RF no era necesario realizarle una Cross Validation para comprobar el error ya que el internamente usa el método de bagging para estimar el error que se obtendrá, por tanto hemos provado a base de ensayo y error con distintas combinaciones de parámetros.

En el caso de la SVM los parámetros en los que nos hemos interesado han sido en la regularización, el kernel y la función de decisión. Los mejores valores se han encontrado con una regularización tipo Lasso y con la función de kernel squared-exponential kernel (RBF) con la siguiente fórmula.

$$k_{SE}(x_i, x_j) = \sigma^2 \exp \frac{(x_i - x_j)^2}{2l^2} \tag{1}$$

donde l es el rango máximo de los datos, y σ determina la distancia entre la media de tu función y la de la población.

Para el caso del RF hemos probado incrementando el número de arboles en la decisión, y las distintas funciones para estimar las variables que tomara cada arbol para ramificar. En el número de arbol vimos que con 50 obteniamos resultados parecidos a usar 100 o 150 pero la capacidad de computó era menor por tanto decidimos que darnos con los 50 arboles. En el máximo de características tomamos $\sqrt{p}$ y $\log p$ encontrando los mejores resultados con el primero de ellos.

5 Resultados