data_wrangling

June 9, 2021

[4]:
```
import numpy as np
import pandas as pd
import joblib
```

[5]:
```
layer_config = []
Precision = []
Recall = []
F1 = []
Cohen_kappa = []
```

[6]:
```
def ReadAndCreate(file):
    results_1 = joblib.load(file)
    layer_config = []
    Precision = []
    Recall = []
    F1 = []
    Cohen_kappa = []
    for i in range(len(results_1)):
        aux = results_1[i]
        layer_config.append(aux['layer config'])
        Precision.append(aux['Metrics'][0])
        Recall.append(aux['Metrics'][1])
        F1.append(aux['Metrics'][2])
        Cohen_kappa.append(aux['Metrics'][3])

    data = {'Hidden layers' : layer_config,
            'Precision' : Precision,
           'Recall' : Recall,
            'F1' : F1,
           'Cohen kappa' : Cohen_kappa}
    data = pd.DataFrame(data)
    return data
```

0.1 Resultados con datos raw

[7]:
```
results_data = ReadAndCreate("results_1_joblib")
results_data.sort_values("Precision", ascending=False).head(6)
```

[7]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
28	[250, 250, 250, 250, 250]	0.825196	0.825196	0.825196	0.804531
29	[250, 250, 250, 250, 250, 250]	0.821810	0.821810	0.821810	0.801151
16	[150, 150, 150, 150, 150]	0.820919	0.820919	0.820919	0.799992
23	[200, 200, 200, 200, 200, 200]	0.819138	0.819138	0.819138	0.797986
17	[150, 150, 150, 150, 150, 150]	0.818425	0.818425	0.818425	0.797329
27	[250, 250, 250, 250]	0.816108	0.816108	0.816108	0.794788

[8]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[8]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
0	[50]	0.556308	0.556308	0.556308	0.501478
6	[100]	0.584105	0.584105	0.584105	0.532926
12	[150]	0.605132	0.605132	0.605132	0.557198
18	[200]	0.628653	0.628653	0.628653	0.584144
24	[250]	0.641661	0.641661	0.641661	0.599029
1	[50, 50]	0.677655	0.677655	0.677655	0.639487

Estudiando los datos con mayor puntuación en Precision descubrimos que los modelos con mayor número de neuronas y capas obtienen buenas puntuaciones, sin llegar al 83%. Inicialmente podríamos esperar que mayor número de neuronas está directamente relacionado con puntuación en metricas, pero vemos que no es así. Configuraciones de capas distintas con números de neuronas por encima de 150 dan lugar a resultados similares. Probablemente, la diferencia en puntuación se deba a la naturaleza aleatoria de las redes neuronales a la hora de inicializar los pesos así como en el algoritmo de minimización. Todos los modelos con características parecidas dan lugar a resultados muy similares.

Por otro lado, en los peores resultados, obtenemos lo esperado. Los modelos de una capa con pocas neuronas clasifican muy mal y mejoran su puntuación en orden creciente de neuronas.

0.2 Resultados datos raw one-hot

[9]:
```
results_data = ReadAndCreate("results_1_onehot_joblib")
results_data.sort_values("Precision", ascending=False).head(6)
```

[9]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
21	[200, 200, 200, 200]	0.830898	0.830898	0.830898	0.811140
26	[250, 250, 250]	0.830185	0.830185	0.830185	0.810440
25	[250, 250]	0.824127	0.824127	0.824127	0.803536
27	[250, 250, 250, 250]	0.817177	0.817177	0.817177	0.795549
22	[200, 200, 200, 200, 200]	0.814505	0.814505	0.814505	0.793240
20	[200, 200, 200]	0.809694	0.809694	0.809694	0.787236

[10]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[10]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
0	[50]	0.713293	0.713293	0.713293	0.679594
5	[50, 50, 50, 50, 50, 50]	0.724519	0.724519	0.724519	0.692311
3	[50, 50, 50, 50]	0.729330	0.729330	0.729330	0.697647
4	[50, 50, 50, 50, 50]	0.732359	0.732359	0.732359	0.701160
1	[50, 50]	0.735923	0.735923	0.735923	0.705121
6	[100]	0.740556	0.740556	0.740556	0.710106

Estudiamos los modelos con mayor puntuación. El uso de usar la codificación *one-hot* de las variables dan lugar a prácticamente los mismos resultados que sin usar esta transformación. Además, los modelos que consiguen las mayores puntuaciones también son muy parecidos.

Sin embargo, en los modelos que obtienen los peores resultados, éstos son muchos mejores que los obtenidos por los peores modelos sin usar *one-hot*. Las peores métricas obtenidas está acotadas inferiormente, en el caso de una sola capa oculta de 50 neuronas, en el 71%. Esto nos puede indicar que, mientras que usar *one-hot* es beneficioso en este tipo de modelos, sea difícil mejorar los resultados obtenidos en los mejores casos.

0.3 Resultados datos raw smote

[11]:
```
results_data = ReadAndCreate("results_smote_joblib")
results_data.sort_values("Precision", ascending=False).head(6)
```

[11]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
29	[250, 250, 250, 250, 250, 250]	0.361725	0.361725	0.361725	0.300894
23	[200, 200, 200, 200, 200, 200]	0.356557	0.356557	0.356557	0.296138
16	[150, 150, 150, 150, 150]	0.355132	0.355132	0.355132	0.295350
27	[250, 250, 250, 250]	0.354775	0.354775	0.354775	0.294756
17	[150, 150, 150, 150, 150, 150]	0.354419	0.354419	0.354419	0.293245
28	[250, 250, 250, 250, 250]	0.349786	0.349786	0.349786	0.288579

[12]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[12]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
0	[50]	0.273521	0.273521	0.273521	0.210563
6	[100]	0.299715	0.299715	0.299715	0.237226
12	[150]	0.300249	0.300249	0.300249	0.238289
1	[50, 50]	0.301675	0.301675	0.301675	0.238587
18	[200]	0.312723	0.312723	0.312723	0.250874
2	[50, 50, 50]	0.319672	0.319672	0.319672	0.257319

En un intento de mejorar el desbalanceo existente en las clases a clasificar, utilizamos el algoritmo SMOTE para mejorar el dataset. El motivo por el que usamos un algoritmo de sobremuestreo en vez de *tirar a la baja* con undersampling es por el beneficio que tienen las redes neuronales al aumentar el tamaño del dataset. Además, en results_datas realizadas con árboles de decisión en otras asignaturas, descubrimos que, a pesar de romper el problema de desbalanceo, obtenemos

peores resultados al disponer de menos casos.

No ha mejorado la clasificación. Los resultados son muy malos. Las mejores redes obtienen métricas alrededor del 35%, mientras que las mejoras obtienen un 27%. Al haber tan poca diferencia entre los mejores y peores resultados, podemos estar seguros que no se debe a un número insuficiente de neuronas y capas.

En los problemas en los que las clases están muy desbalanceadas, SMOTE no funciona bien. Esta es una de esas ocasiones. Además, este algoritmo funciona mejor en problemas de clasificación binarios, donde fue concebido.

Como hemos mencionado antes, SMOTE está diseñado para problemas de predictores continuos, donde es más natural la interpolación. Sin embargo, como vimos que no daba ningún error al ejecutar la librería, vimos interesante ver el desempeño de la misma en problemas categóricos.

0.4 Resultados datos smote one-hot

[13]:
```
results_data = ReadAndCreate("results_smote_onehot_joblib")
results_data.sort_values("Precision", ascending=False).head(6)
```

[13]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
19	[200, 200]	0.187634	0.187634	0.187634	0.144296
26	[250, 250, 250]	0.177833	0.177833	0.177833	0.131770
20	[200, 200, 200]	0.152174	0.152174	0.152174	0.107797
25	[250, 250]	0.145937	0.145937	0.145937	0.103981
14	[150, 150, 150]	0.138097	0.138097	0.138097	0.091480
13	[150, 150]	0.129187	0.129187	0.129187	0.091482

[14]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[14]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
5	[50, 50, 50, 50, 50, 50]	0.054348	0.054348	0.054348	0.018471
17	[150, 150, 150, 150, 150, 150]	0.062545	0.062545	0.062545	0.026222
16	[150, 150, 150, 150, 150]	0.066999	0.066999	0.066999	0.031643
4	[50, 50, 50, 50, 50]	0.067890	0.067890	0.067890	0.028956
1	[50, 50]	0.069316	0.069316	0.069316	0.037007
2	[50, 50, 50]	0.069672	0.069672	0.069672	0.034412

La codificación one-hot ha empeorado los resultados con SMOTE. Las métricas están acotadas entre un 18% y un 5%.

1 Con dropout

2 Resultados datos raw

[15]:
```
results_data = ReadAndCreate("results_dropout")
results_data.sort_values("Precision", ascending=False).head(6)
```

[15]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
94	[250, 0.1, 250, 0.1, 250, 0.1, 250, 0.1, 250, …	0.874020	0.874020	0.874020	0.859249
76	[200, 0.1, 200, 0.1, 200, 0.1, 200, 0.1, 200, …	0.873664	0.873664	0.873664	0.858904
95	[250, 0.1, 250, 0.1, 250, 0.1, 250, 0.1, 250, …	0.871347	0.871347	0.871347	0.856290
74	[200, 0.1, 200, 0.1, 200, 0.1]	0.870991	0.870991	0.870991	0.855863
75	[200, 0.1, 200, 0.1, 200, 0.1, 200, 0.1]	0.866180	0.866180	0.866180	0.850478
77	[200, 0.1, 200, 0.1, 200, 0.1, 200, 0.1, 200, …	0.864754	0.864754	0.864754	0.848903

[16]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[16]:

	Hidden layers	Precision	Recall	F1	Cohen kappa
30	[50, 0.3]	0.511048	0.511048	0.511048	0.449370
24	[50, 0.2]	0.517284	0.517284	0.517284	0.456681
35	[50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, …	0.522630	0.522630	0.522630	0.463069
34	[50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]	0.526372	0.526372	0.526372	0.466761
0	[50, 0.1]	0.527263	0.527263	0.527263	0.469118
18	[50, 0.1]	0.530649	0.530649	0.530649	0.472995

Los mejores resultados mejoran levemente los obtenidos sin dropout. Al igual que en estos últimos, las modelos más grandes obtienen mejores resultados. Notemos que todos los que aparecen con mejores métricas tienen un porcentaje de desactivación muy pequeño.

En el polo opuesto, los modelos que obtuvieron peor resultado fueron los modelos con el mayor porcentaje de desactivación en dropout: 30% y menor número de neuronas. Esto es lógico, ya que no parece ser un dataset lo suficientemente grande como para que se produzca un sobreaprendizaje durante el entrenamiento.

Recordamos que el número de épocas en el entrenamiento viene controlado por tensorflow mediante los *Callbacks*, como comentamos anteriormente, por lo que si no mejora la *accuracy* del conjunto de validación en 10 épocas, finaliza el entrenamiento.

2.1 Resultados datos raw one-hot

[17]:
```
results_data = ReadAndCreate("results_dropout_one_hot")
results_data.sort_values("Precision", ascending=False).head(6)
```

[17]:
```
                                        Hidden layers  Precision    Recall  \
77  [250, 0.1, 250, 0.1, 250, 0.1, 250, 0.1, 250, …   0.894512  0.894512
62                     [200, 0.2, 200, 0.2, 200, 0.2]   0.892908  0.892908
79                               [250, 0.2, 250, 0.2]   0.891839  0.891839
63           [200, 0.2, 200, 0.2, 200, 0.2, 200, 0.2]   0.888988  0.888988
56                     [200, 0.1, 200, 0.1, 200, 0.1]   0.888453  0.888453
80                     [250, 0.2, 250, 0.2, 250, 0.2]   0.887741  0.887741

          F1  Cohen kappa
77  0.894512     0.882191
62  0.892908     0.880426
79  0.891839     0.879262
63  0.888988     0.876026
56  0.888453     0.875431
80  0.887741     0.874633
```

[18]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[18]:
```
                                        Hidden layers  Precision    Recall  \
17  [50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, …   0.607627  0.607627
16      [50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]   0.640770  0.640770
15               [50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]   0.655560  0.655560
12                                          [50, 0.3]   0.658411  0.658411
14                       [50, 0.3, 50, 0.3, 50, 0.3]   0.669280  0.669280
6                                           [50, 0.2]   0.679793  0.679793

          F1  Cohen kappa
17  0.607627     0.561731
16  0.640770     0.597454
15  0.655560     0.614280
12  0.658411     0.617240
14  0.669280     0.629326
6   0.679793     0.641515
```

Los resultados son esperables vistos los casos anteriores. Dropout mejora también el caso en el que usamos *one-hot* encoding. Estos son los mejores resultados que obtenemos.

2.2 Resultados datos raw smote

[19]:
```
results_data = ReadAndCreate("results_dropout_smote")
results_data.sort_values("Precision", ascending=False).head(6)
```

[19]:
```
                                        Hidden layers  Precision    Recall  \
62                      [200, 0.2, 200, 0.2, 200, 0.2]   0.403243  0.403243
58  [200, 0.1, 200, 0.1, 200, 0.1, 200, 0.1, 200, …   0.400214  0.400214
77  [250, 0.1, 250, 0.1, 250, 0.1, 250, 0.1, 250, …   0.400036  0.400036
83  [250, 0.2, 250, 0.2, 250, 0.2, 250, 0.2, 250, …   0.399501  0.399501
80                      [250, 0.2, 250, 0.2, 250, 0.2]   0.399501  0.399501
74                      [250, 0.1, 250, 0.1, 250, 0.1]   0.398610  0.398610

          F1  Cohen kappa
62  0.403243     0.346788
58  0.400214     0.342223
77  0.400036     0.341473
83  0.399501     0.341194
80  0.399501     0.342195
74  0.398610     0.341175
```

[20]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[20]:
```
                                        Hidden layers  Precision    Recall  \
17  [50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, …   0.215966  0.215966
16      [50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]   0.227548  0.227548
15               [50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]   0.243763  0.243763
12                                          [50, 0.3]   0.245902  0.245902
6                                           [50, 0.2]   0.249465  0.249465
48                                         [150, 0.3]   0.254098  0.254098

          F1  Cohen kappa
17  0.215966     0.151163
16  0.227548     0.161465
15  0.243763     0.179339
12  0.245902     0.185292
6   0.249465     0.187577
48  0.254098     0.189198
```

Se obtienen mejores resultados que sin incluir dropout, pero siguen siendo malos.

2.3 Resultados datos smote one-hot

[21]:
```
results_data = ReadAndCreate("results_dropout_smote_one_hot")
results_data.sort_values("Precision", ascending=False).head(6)
```

[21]:
```
            Hidden layers  Precision    Recall        F1  Cohen kappa
55  [200, 0.1, 200, 0.1]   0.370813  0.370813  0.370813     0.321053
79  [250, 0.2, 250, 0.2]   0.367071  0.367071  0.367071     0.316297
73  [250, 0.1, 250, 0.1]   0.355845  0.355845  0.355845     0.306629
37  [150, 0.1, 150, 0.1]   0.314505  0.314505  0.314505     0.261605
84            [250, 0.3]   0.313257  0.313257  0.313257     0.252906
```

```
78            [250, 0.2]   0.296686  0.296686  0.296686     0.238117
```

[22]:
```
results_data.sort_values("Precision", ascending=True).head(6)
```

[22]:
```
                                        Hidden layers  Precision    Recall  \
16      [50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]   0.019601  0.019601
15                [50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3]   0.023521  0.023521
17  [50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, 50, 0.3, …   0.026907  0.026907
9                 [50, 0.2, 50, 0.2, 50, 0.2, 50, 0.2]   0.031896  0.031896
10      [50, 0.2, 50, 0.2, 50, 0.2, 50, 0.2, 50, 0.2]   0.033856  0.033856
5   [50, 0.1, 50, 0.1, 50, 0.1, 50, 0.1, 50, 0.1, …   0.034569  0.034569

          F1  Cohen kappa
16  0.019601     0.003531
15  0.023521    -0.000884
17  0.026907     0.009679
9   0.031896     0.003915
10  0.033856     0.009683
5   0.034569     0.008312
```

Se repite el comportamiento, usar dropout con un porcentaje de desactivación pequeño mejora los resultados, pero éstos eran muy malos per se.

2.4 Conclusiones

Hemos construido modelos de redes neuronales del tipo perceptrón multicapa con distintas configuraciones de capas ocultas y neuronas. Los datos utilizados han sido los originales, la codificación de las variables predictoras mediante *one-hot* y sus equivalentes tras usar *SMOTE*. También se ha implementado *callbacks* para parar el entrenamiento si el valor de *accuracy* en el set de validación no mejora tras 10 épocas, evitando el sobreaprendizaje y retirando el número de épocas como parámetro a estudiar y modificar. A todos estos modelos, se hicieron experimentos introduciendo capas intermedias con *dropout* con tres posibles valores de desactivación de neuronas: 10, 20 y 30%.

Los resultados son los siguientes. El uso de *SMOTE* no aporta ningún beneficio. Las métricas obtenidas son mucho peores que usando datos sin *SMOTE*. La razón puede deberse al gran desbalanceo entre clases que existe en la variable a predecir y la naturaleza categórica de las predictoras. El algoritmo fue diseñado y funciona mejor en problemas binarios con entradas continuas donde la interpolación parece más natural.

Introducir *dropout* mejora el resultado en todos los casos, incluso en los peores.

Creemos haber llegado a resultados próximos a los máximos posibles usando perceptrones multicapa ya que, en los casos con mejor puntuación, no existe una relación directa entre mayor número de neuronas y rendimiento. Se dan casos en los que configuraciones con menos capas obtienen mejores resultados que equivalentes con más. Sin embargo, está claro que un número grande neuronas es beneficioso para el modelo.