APRENDIZAJE AUTOMÁTICO AVANZADO

INFORME TÉCNICO UNIDAD II – APRENDIZAJE SUPERVISADO

PRESENTADO POR:

Edgar Leandro Jiménez Jaimes

Santiago Echeverri Calderón

DOCENTE:

José Lisandro Aguilar Castro

UNIVERSIDAD EAFIT

MEDELLÍN

MAESTRÍA EN CIENCIAS DE LOS DATOS Y ANALÍTICA

FEBRERO DE 2020

1. Objetivo de la iteración

Analizar y aplicar en un conjunto de datos diferentes técnicas de Aprendizaje Supervisado para la construcción de modelos de clasificaciones binaria, donde finalmente se busca medir el desempeño de los modelos mediante diferentes métricas.

2. Contextualización del problema

Las técnicas de Aprendizaje Supervisado se implementarán en un conjunto de datos de la Guía de Campo de la Sociedad Audobon; son datos de hongos descritos en términos de características físicas; y se tiene su respectiva clasificación: venenoso o comestible

"Este conjunto de datos incluye descripciones de muestras hipotéticas correspondientes a 23 especies de hongos branquiales de la familia Agaricus y Lepiota (pp. 500-525). Cada especie se identifica como definitivamente comestible, definitivamente venenosa o de comestibilidad desconocida y no se recomienda. Esta última clase se combinó con la venenosa. La Guía establece claramente que no existe una regla simple para determinar la comestibilidad de un hongo."

Presentamos a continuación la información de los atributos que contiene el conjunto de datos: Información del atributo:

1. **Forma De Tapa:** campana = b, cónica = c, convexa = x, plana = f, nudosa = k, hundida = s
2. **Superficie De Tapa:** fibrosa = f, ranuras = g, escamosa = y, lisa = s
3. color de la tapa: marrón = n, pulido = b, canela = c, gris = g, verde = r, rosa = p, púrpura = u, rojo = e, blanco = w, amarillo = y
4. **Moretones:** moretones = t, no = f
5. **Olor:** almendra = a, anís = l, creosota = c, a pescado = y, asqueroso = f, a humedad = m, ninguno = n, picante = p, picante = s
6. **Branquias- Archivo Adjunto**: adjunto = a, descendente = d, libre = f, con muesca = n
7. **Espaciado De Branquias**: cerca = c, abarrotado = w, distante = d
8. **Tamaño De Las Branquias**: ancho = b, estrecho = n
9. **Color De La Branquia**: negro = k, marrón = n, pulido = b, chocolate = h, gris = g, verde = r, naranja = o, rosa = p, púrpura = u, rojo = e, blanco = w, amarillo = y

10. **Forma De Tallo**: agrandando = e, estrechando = t
11. **Raíz De Tallo**: bulbosa = b, garrote = c, copa = u, igual = e, rizomorfos = z, enraizada = r, perdida =?
12. **Superficie Del Tallo Por Encima Del Anillo**: fibroso = f, escamoso = y, sedoso = k, liso = s
13. **Superficie Del Tallo Por Debajo Del Anillo**: fibroso = f, escamoso = y, sedoso = k, liso = s
14. **Color Del Tallo Encima Del Anillo**: marrón = n, piel de ante = b, canela = c, gris = g, naranja = o, rosa = p, rojo = e, blanco = w, amarillo = y
15. **Color De Tallo -Abajo-Anillo:** marrón = n, pulido = b, canela = c, gris = g, naranja = o, rosa = p, rojo = e, blanco = w, amarillo = y
16. **Tipo De Velo**: parcial = p, universal = u
17. **Color Del Velo**: marrón = n, naranja = o, blanco = w, amarillo = y
18. **Número De Anillo:** ninguno = n, uno = o, dos = t
19. **Tipo De Anillo**: telaraña = c, evanescente = e, quemado = f, grande = l, ninguno = n, colgante = p, revestimiento = s, zona = z
20. **Color De Impresión De Esporas**: negro = k, marrón = n, pulido = b, chocolate = h, verde = r, naranja = o, púrpura = u, blanco = w, amarillo = y
21. **Población**: abundante = a, agrupada = c, numerosa = n, dispersa = s, varias = v, solitaria = y
22. **Hábitat**: pastos = g, hojas = l, prados = m, caminos = p, urbanos = u, desechos = w, bosques = d

Como se puede notar, todos los atributos del dataset con categóricos, motivo especial por el cual elegimos trabajar con este conjunto de datos, ya que hoy por hoy las organizaciones y sus datos contienen mucha información de este tipo, con lo cual quisimos enfocar esfuerzo en aprender como trabajarlo y modelarlo en aprendizaje supervisado

El conjunto de datos se encuentra disponible en:

https://www.kaggle.com/uciml/mushroom-classification

3. Diseño del modelo

El modelo consta de las siguientes etapas:

i. Exploración y preprocesamiento de los datos.
ii. Implementación de tres modelos de clasificación binaria.
iii. Evaluación de desempeño de los modelos.

3.1. Exploración y preprocesamiento de los datos.

El dataset original consta entonces de 22 variables categóricas y una variable binaria que indica la clasificación: venenoso o comestible. En total se cuenta con 8124 observaciones.

Realizamos una rápida exploración de los datos de las variables categóricas:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	stalk-root	stalk-surface-above-ring	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
count	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124
unique	2	6	4	10	2	9	2	2	2	12	2	5	4	4	9	9	1	4	3	5	9	6	7
top	e	x	y	n	f	n	f	c	b	b	t	b	s	s	w	w	p	w	o	p	w	v	d
freq	4208	3656	3244	2284	4748	3528	7914	6812	5612	1728	4608	3776	5176	4936	4464	4384	8124	7924	7488	3968	2388	4040	3148

Podemos observar que el atributo veil-type solo tienen una única clase, con lo cual procedemos a retirarla porque no aporta información relevante.

Posteriormente procedemos a buscar si existen valores nulos o faltantes en los datos. Como se muestra a continuación, la búsqueda no arroja valores faltantes para ningún atributo.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 22 columns):
class                     8124 non-null object
cap-shape                 8124 non-null object
cap-surface               8124 non-null object
cap-color                 8124 non-null object
bruises                   8124 non-null object
odor                      8124 non-null object
gill-attachment           8124 non-null object
gill-spacing              8124 non-null object
gill-size                 8124 non-null object
gill-color                8124 non-null object
stalk-shape               8124 non-null object
stalk-root                8124 non-null object
stalk-surface-above-ring  8124 non-null object
stalk-surface-below-ring  8124 non-null object
stalk-color-above-ring    8124 non-null object
stalk-color-below-ring    8124 non-null object
veil-color                8124 non-null object
ring-number               8124 non-null object
ring-type                 8124 non-null object
spore-print-color         8124 non-null object
population                8124 non-null object
habitat                   8124 non-null object
dtypes: object(22)
memory usage: 1.4+ MB
```

Sin embargo, al leer acerca del dataset encontramos que si hay registros faltantes, pero estos son datos son llenado mediante “?”. Observemos:

```
class                        0
cap-shape                    0
cap-surface                  0
cap-color                    0
bruises                      0
odor                         0
gill-attachment              0
gill-spacing                 0
gill-size                    0
gill-color                   0
stalk-shape                  0
stalk-root                2480
stalk-surface-above-ring     0
stalk-surface-below-ring     0
stalk-color-above-ring       0
stalk-color-below-ring       0
veil-color                   0
ring-number                  0
ring-type                    0
spore-print-color            0
population                   0
habitat                      0
dtype: int64
```

En realidad, para el atributo stalk-root se encontró 2480 registros con “?”. Para estos registros encontramos en los diferentes blogs y competencias de kaggle que fueron recategorizados como una nueva clase.

Como se dijo anteriormente, la motivación principal para trabajar con este dataset es poder trabajar con variables categóricas. Nos dimos a la tarea de buscar sobre esta modelación y encontramos dos enfoques comúnmente utilizados: LabelEncoding y OneHotEncoding.

Revisamos las discusiones acerca de utilizar el LabelEnconing y el OneHotEncoding y se concluyó en trabajar con el LabelEnconing, ya que estamos realizando una tarea de clasificación y uno de los algoritmos que utilizaremos serán los árboles de clasificación, el cual, según algunas discusiones que revisamos no es conveniente utilizar el OneHotEncoding.[1]

Entonces vamos a realizar un LabelEncoding para las variables categóricas. Este genera un numero entero para cada categoría de la columna. Presentamos a continuación una muestra de cómo queda el dataset luego de aplicar el Label.

[1] Discusión sobre OneHotEncoding en Arboles de clasificación:
https://roamanalytics.com/2016/10/28/are-categorical-variables-getting-lost-in-your-random-forests/

	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	stalk-root	stalk-surface-above-ring
0	5	2	4	1	6	1	0	1	4	0	2	2
1	5	2	9	1	0	1	0	0	4	0	1	2
2	0	2	8	1	3	1	0	0	5	0	1	2

Posterior a esto realizamos un proceso de estandarización de los datos antes de aplicar cualquier modelo supervisado. Para esta tarea utilizamos un StandarScaler que es calculada como $z = (x - \mu) / s$.

En un problema de clasificación binaria es importante tener en cuenta el balanceo de clases, para esto vamos a contar cuentas clases hay de cada una:

```
e    0.517971
p    0.482029
```

Es decir que modelo base tendrá un desempeño de 51,79%, es decir, no hacer "nada" es tener ese desempeño de modelo. Por lo cual esperamos que al utilizar modelos matemáticos y estadísticos podamos mejorar el desempeño.

Ahora bien, para realizar la modelación de los algoritmos supervisados, utilizamos una partición de la siguiente manera:

- Para el conjunto de entrenamiento de los modelos vamos a utilizar el 60% de los datos.
- Para validación de los modelos vamos a utilizar 20%.
- Y para realizar el test final empleamos un 20%.

3.2. Implementación de algoritmos y resultado preliminares

Ahora vamos a realizar la modelación de datos. Para esto vamos a realizar tres modelos de clasificación supervisada:

- Regresión Logística + Regularización
- Máquinas de soporte vectorial
- Random Forest

Regresión logística + Regularización

El primer método que se implementó fue el modelo de regresión logística. Este modelo de regresión se utilizó con los parámetros por defecto que trae la librería, para posteriormente realizar procesos de optimización de hiper-parámetros, tanto para evitar el sobre ajuste como para mejorar la precisión del modelo.
En esta primera corrida el modelo obtuvo lo siguiente desempeño:

- En la regresión logística en train el accuracy es de 0.964
- En la regresión logística en validación el accuracy es de 0.961
- En la regresión logística en test el accuracy es de 0.954

Posteriormente se realiza una optimización de hiperparametros en el cual se habilita la regularización, a continuación, se presenta la configuración:

```
parameters={"C":np.logspace(-3,3,7),
            "penalty":["l1","l2"]}
```

El resultado obtenido a partir de esta optimización y al aplicar un K-Cross Validation con K = 5 son los siguientes:

- El desempeño de la regresión logística en test luego de regularizar es de 0.963.

Observamos entonces que el resultado es ligeramente mejor en el conjunto de datos de testeo que aplicamos. En la siguiente sección se ampliará mas los resultados utilizando otras métricas.

Máquinas de soporte vectorial

El segundo método que se implementó fue una máquina de soporte vectorial, el cual es un modelo más sofisticado y robusto que la regresión logística. Al igual que para el primer modelo, la primera corrida se utilizó con los parámetros por defecto que trae la librería, para posteriormente realizar procesos de optimización de hiperparametros, tanto para evitar el sobre ajuste como para mejorar la precisión del modelo.

En esta primera corrida el modelo obtuvo el siguiente siguiente:

- El accuracy del modelo SVM en train es de 1.0

- El accuracy del modelo SVM en validación es de 0.998
- El accuracy del modelo SVM en test es de 0.999

Posteriormente se realiza una optimización de hiperparametros, a continuación, se presenta la configuración:

```
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
'C':[1, 10, 20, 30]},
{'kernel': ['linear'], 'C':[1, 10, 20, 30]}]
```

El resultado obtenido a partir de esta optimización y al aplicar un K-Cross Validation con K = 5 son los siguientes:

- El accuracy del modelo SVM en train es de 0.979
- El accuracy del modelo SVM en validación es de 0.977
- El accuracy del modelo SVM en test es de 0.973

Observamos entonces que el resultado en estas métricas disminuye un poco, lo cual se debe al efecto de la búsqueda y optimización de los hiperparametros, recordemos que lo que se busca es que el modelo generalice sobre los datos y no que se los aprenda de memoria.

Random Forest (O bosques aleatorios)

El ultimo método que se implementó fue un random forest. Al igual que para el primer modelo, la primera corrida se utilizó con los parámetros por defecto que trae la librería, para posteriormente realizar procesos de optimización de hiperparametros, tanto para evitar el sobre ajuste como para mejorar la precisión del modelo.

En esta primera corrida el modelo obtuvo lo siguiente:

- El accuracy del random forest en train es de 0.991
- El accuracy del random forest en validacion es de 0.991
- El accuracy del random forset en test es de 0.994

Posteriormente se realiza una optimización de hiperparametros, a continuación, se presenta la configuración:

```
param_grid = {'n_estimators': [150,250,300
'max_features': ['sqrt', 'log2'],
'max_depth':[5,8,10],
'criterion':['gini', 'entropy']}
```

El resultado obtenido a partir de esta optimización y al aplicar un K-Cross Validation con K = 5 son los siguientes:

- El accuracy del random en train es de 1.0
- El accuracy del random en validacion es de 1.0
- El accuracy del random en test es de 1.0

Este resultado, y en general los modelos propuestos, presentan rendimientos considerablemente altos respecto a la línea base que se trazo por el balance natural del dataset. Una hipótesis es que el conjunto de datos es fácilmente separable, lo que hace que un modelo como Ramdon Forest sea capaz de dividir perfectamente los datos en clase 0 y clase 1. Esto teniendo en cuenta que los datos son categóricos pueden marcar un fuerte patrón que permite que la división sea sencilla para modelos mas sofisticados.

4. Análisis de métricas y comparación de modelos

Observemos los resultados obtenidos con métricas de Recall, Precision y F1 Score, que son métricas mas adecuadas cuando se esta trabajando en problemas de clasificación binaria.

	Precisión	Recall	F1-Score
Clase 0	0,97	0,95	0,96
Clase 1	0,96	0,97	0,96
Accuracy			0,96

Regresión logística optimizada

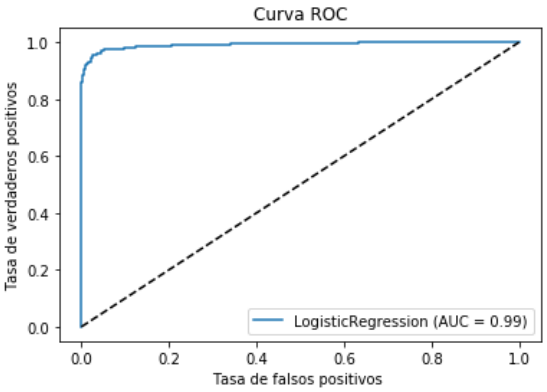
	Precisión	Recall	F1-Score
Clase 0	0,97	0,99	0,98
Clase 1	0,99	0,97	0,98
Accuracy			0,98

Máquina de soporte vectorial optimizada

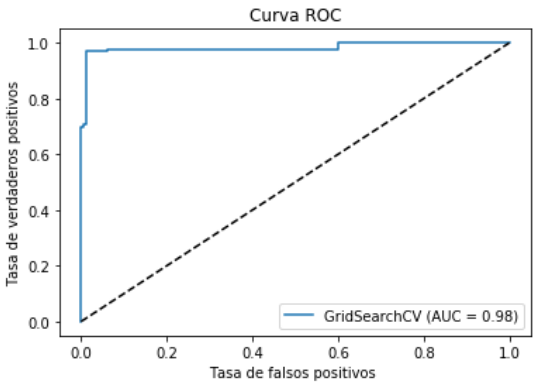
	Precisión	Recall	F1-Score
Clase 0	1	1	1
Clase 1	1	1	1
Accuracy			1

Random Forest optimizado

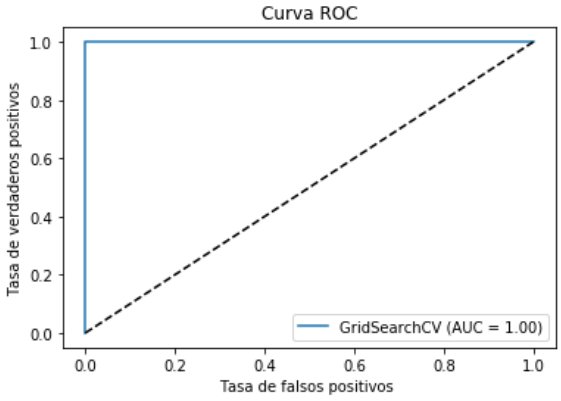
Ahora vamos a presentar las curvas ROC y el AUC para los tres modelos:



Regresión logística optimizada



Máquina de soporte vectorial optimizada



Random Forest optimizado

Observamos entonces que los tres métodos tienen desempeños muy altos en todas las métricas que se estudiaron, desde la más básica que es el accuracy como métricas más elaboradas como la F1-score, así mismo observamos que las curva ROC y los valores AUC (Area Under Curve) de los tres modelos son muy buenas. Como se mencionó anteriormente, parece ser que este es un problema con una línea muy marcada de separación, lo que hace que estos modelos, desde el mas sencillo como la regresión hasta un random forest logren capturar esta separación y generalicen los patrones, lo que lleva a que se obtengan desempeños altos en los conjuntos de validación y prueba.

5. Cuaderno Jupyter

El Notebook con la implementación puede encontrarse en:

https://github.com/santiagoec/CM0891-Aprendizaje-Automatico/blob/master/02_Aprendizaje_Supervisado.ipynb