

3-11-2020

Práctica 1: Problema de clasificación

Inteligencia de negocio



Juan Carlos González Quesada
UGR

Índice

Introducción	3
Procesado de datos.....	3
Primer análisis.....	3
Segundo análisis	4
Configuración de Algoritmos.....	4
Árboles de decisión:.....	4
Support Vector Machines	5
Ensembled method	7
Redes neuronales.....	8
Resultados obtenidos	9
Nayve-Bayes	9
Análisis de resultados Nayve-Bayes:.....	10
Árboles de decisión	11
Análisis de resultados Árboles de decisión:.....	11
Support Vector Machines	12
Análisis de resultados SVM:.....	13
Ensembled methods.....	13
Análisis de resultados Ensembled Methods:	14
Redes neuronales.....	14
Análisis de resultados Redes Neuronales:.....	15
Análisis de resultados	15
Caso 1: Eliminando valores perdidos y sin parámetros.....	15
Caso 2: Modificando valores y sin parámetros	16
Caso 3: Modificando y eliminando valores y con parámetros	17
Con parámetros y modificando los datos	17
Con parámetros y eliminando los valores perdidos.....	18
¿Hay algoritmos mejores?.....	19
Nayve-Bayes	20
Árboles de decisión	20
Support Vector Machine.....	20
Ensembled Methods.....	20
Redes neuronales.....	20
Interpretación general	21

Interpretación de resultados.....	22
Nayve-Bayes	22
Árboles de decisión	22
Support Vector Machine.....	23
Ensembled methods.....	23
Redes neuronales.....	24
Contenido adicional.....	25
Bibliografía	26

Introducción

Dado un conjunto de datos sobre mamografías, se pide usar algoritmos de clasificación para realizar un análisis determinando cuales de los casos tendrán tumores malignos o benignos, estudiando su comportamiento.

Para la correcta realización, trabajaremos, durante el desarrollo de la práctica, con cinco algoritmos de clasificación: Naive-Bayes, árboles de decisión, SVM, ensembled methods y redes neuronales. Además, estudiaremos cómo se comportan estos algoritmos de aprendizaje con distintas características.

Procesado de datos

Primer análisis

Para el primer análisis, borraremos las filas que contienen los valores ‘perdidos’. De esta forma, vemos un claro inconveniente, y es que se disminuye en 300 filas (aproximadamente) el conjunto de datos sobre los que se realiza el estudio. Sin embargo, obtenemos unos datos que son fiables, puesto que no estarán modificados.

```
mamografias= pd.read_csv("./mamografias.csv",na_values=[ "?" ])
mamografias=mamografias.dropna()
```

Para evitar problemas de compatibilidad de tipos, se van a codificar las columnas de Shape y Severity

```
mamografias['Shape']=le.fit_transform(mamografias['Shape'])
mamografias['Severity']=le.fit_transform(mamografias['Severity'])
```

Para los conjuntos de entrenamiento y test, se realizará la siguiente partición:

```
data_train, data_test, target_train, target_test = train_test_split(atributos ,target, test_size = 0.8, random_state = 5)
```

Segundo análisis

Para el segundo análisis, no eliminaremos los datos, por el contrario, efectuaremos un parseado de los datos. El valor perdido tomará el del que se encuentre con más frecuencia.

```
mamografias= pd.read_csv("./mamografias.csv",na_values=["?"])
mamografias['Density']=mamografias['Density'].fillna(mode(mamografias['Density']))
mamografias['BI-RADS']=mamografias['BI-RADS'].fillna(mode(mamografias['BI-RADS']))
mamografias['Margin']=mamografias['Margin'].fillna(mode(mamografias['Margin']))
mamografias['Age']=mamografias['Age'].fillna(mode(mamografias['Age']))
mamografias['Shape']=mamografias['Shape'].fillna(mode(mamografias['Shape']))
```

Configuración de Algoritmos

Hemos buscado en la página de *Scikit-Learn* algunos de los algoritmos que pueden ser modificados al añadirle parámetros. A continuación, mostraremos su estudio.

Para Nayve-Bayes, no se podrá realizar el estudio con parámetros puesto que este algoritmo no los admite.

Árboles de decisión:

En ambos se añaden los mismos parámetros:

random_state: Controla la mezcla aplicada a los datos antes de aplicar la división. El valor que se le asigna es para una salida reproducible a través de múltiples llamadas a funciones.

max_depth: Para el nivel de profundidad

- Normal:

```
arbNor = tree.DecisionTreeClassifier(random_state=4, max_depth=2)
arbNor = arbNor.fit(data_train, target_train)
predADnor = arbNor.predict(data_test)
scoresADnor = cross_val_score(arbNor, atributos, target, cv=5,
scoring='accuracy')
```

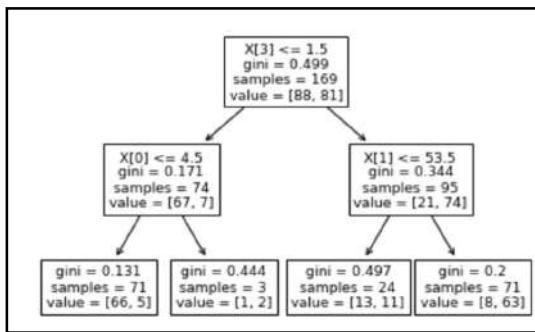
- Extra:

```
arbEx = tree.ExtraTreeClassifier(random_state=2, max_depth=3)
arbEx = arbEx.fit(data_train, target_train)
predADEX = arbEx.predict(data_test)
scoresADEX = cross_val_score(arbEx, atributos, target, cv=5,
scoring='accuracy')
```

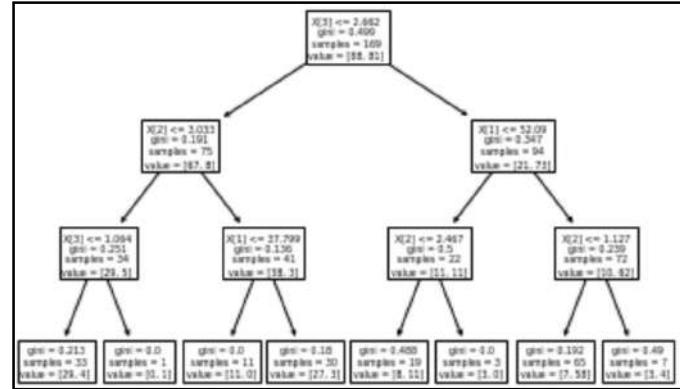
Ahora, obtenemos los porcentajes y dibujamos de nuevo los árboles:

Tipo	Acierto Mejora (%)	Acierto Inicial (%)
Normal	81.68	74.96
Extra	76.38	75.08

Normal:



Extra:



Observamos como hay una mejora muy notable para el primer algoritmo, colocándose, por ahora, como el mejor en todas las pruebas, en cambio, para el extra, hay una reducción de su tasa de acierto.

¿Qué pasaría si con las mejoras de los parámetros, usamos el primer tratamiento de los datos, es decir, en vez de parsear los valores perdidos, eliminamos las filas?

Tipo	Acierto Mejora (%)	Acierto Mejora + eliminar datos (%)
Normal	81.68	82.04
Extra	76.38	77.21

Hay una nueva mejora, aunque esta vez pequeña, en el algoritmo de clasificación normal y una gran mejora en el extra. Por tanto, podemos discernir que parsear los datos afecta al acierto en la predicción. Desarrollaremos esta cuestión en el siguiente punto.

Support Vector Machines

En ambos algoritmos añadimos:

Max_iter: que indica el número máximo de iteraciones que realizará.

Y en NuSVC añadiremos el parámetro **random_state**, explicado anteriormente.

- NuSVC

```
svr_nu = NuSVC(random_state=10,max_iter=3000)
svr_nu.fit(data_train, target_train)
predsvNu = svr_nu.predict(data_test)
scoresNu = cross_val_score(svr_nu, atributos, target, cv=5, scoring='accuracy')
```

- SVC

```
svr_svc = SVC(max_iter=3000)
svr_svc.fit(data_train, target_train)
predsvSvc = svr_svc.predict(data_test)
scoresSvc = cross_val_score(svr_svc, atributos, target, cv=5, scoring='accuracy')
```

Los nuevos porcentajes obtenidos son:

Tipo	Acierto Mejora (%)	Acierto Inicial (%)
NuSVC	81.27	80.52
SVC	79.60	78.15

En la tabla, se observa que los resultados se mejoran en ambos casos, pero no de una forma significativa. Si la ejecución se hiciera eliminando los datos, ¿qué pasaría?

Para nuestra sorpresa, obtenemos el mismo porcentaje de acierto que al principio. Podemos decir que la mejora de los algoritmos con los parámetros no tiene efecto y en este caso, ha servido realizar el parseado de los datos.

Tipo	Acierto Mejora (%)	Acierto Mejora + eliminar datos (%)
NuSVC	81.27	80.52
SVC	79.60	78.15

Ensembled method

Explicamos de forma individual los parámetros, debido a que son diferentes para cada uno.

- Bagging meta-estimator:
 - **KNeighborsClassifier**: Clasificador que implementa el voto de los k vecinos más cercanos.
 - **Max_samples**: El número de muestras a extraer de X para entrenar a cada estimador base.
 - **Max_features**: El número de características que se deben extraer de X para entrenar cada estimador base.

```
bagging = BaggingClassifier(KNeighborsClassifier(), max_samples=0.5, max_features=0.5)
bagging.fit(data_train, target_train)
preBag = bagging.predict(data_test)
scoresBag = cross_val_score(bagging, atributos, target, cv=5, scoring='accuracy')
```

- Random Forest
 - **N_estimators**: El número de estimadores bases en el conjunto.
 - **Max_depth**: máxima profundidad.
 - **Min_samples_split**: El número mínimo de muestras necesarias para dividir un nodo interno
 - **Random_state**: Controla el muestreo aleatorio del conjunto de datos original.

```
baforests = RandomForestClassifier(n_estimators=10, max_depth=None,min_samples_split=2, random_state=0)
forests.fit(data_train, target_train)
preFo = forests.predict(data_test)
scoresFo = cross_val_score(forests, atributos, target, cv=5, scoring='accuracy')
```

Los nuevos porcentajes obtenidos son:

Tipo	Acierto Mejora (%)	Acierto Inicial (%)
Bagging	83.14	77.56
R. Forest	79.81	78.50

Al realizar los cambios, obtenemos un buen resultado y una gran mejora en el primer algoritmo. ¿Qué ocurriría si eliminamos los datos?

Tipo	Acierto Mejora (%)	Acierto Mejora + eliminar datos (%)
Bagging	83.14	83.00
R. Forest	79.81	78.03

Aunque en el Random Forest hay una disminución de la tasa de acierto, en el Bagging obtenemos la mayor tasa de acierto encontrada hasta ahora. Por tanto, de nuevo comprobamos que eliminando los datos o procesándolos obtenemos las mejores tasas, aunque esta vez por 0,14 décimas, es mejor procesar los datos.

Redes neuronales

Explicamos de forma individual los parámetros, debido a que son diferentes para cada uno.

- MLPClassifier:
 - **Activation:** función de activación para la capa oculta. Usaré la función hiperbólica ‘tanh’
 - **Mas_iter:** máximo número de iteraciones.

```
modelMLP = MLPClassifier(activation='tanh', max_iter=9000)
modelMLP.fit(data_test, target_test)
scores = cross_val_score(modelMLP, atributos, target, cv=5, scoring='accuracy')
salida=modelMLP.predict(data_test)
```

- KNC:
 - **N_neighbours:** Número de vecinos que se utilizarán de forma predeterminada para las k consultas.

```
KNC = KNeighborsClassifier(n_neighbors= 2)
KNC.fit(data_test,target_test)
salida2=KNC.predict(data_test)
score2 = cross_val_score(KNC, atributos, target, cv=5, scoring='accuracy')
```

Los nuevos porcentajes obtenidos son:

Tipo	Acierto Mejora (%)	Acierto Inicial (%)
MLPClassifier	81.06	80.04
KNC	74.08	80.03

Aunque obtenemos una mejora para el tipo MLPClassifier, conseguimos un notable descenso de acierto en el KNC. Con este cambio, se nota más la diferencia de acierto entre los algoritmos, que al principio parecían tener la misma tasa de acierto. ¿Ocurrirá lo mismo al eliminar los datos?

Tipo	Acierto Mejora (%)	Acierto Mejora + eliminar datos (%)
MLPClassifier	81.06	81.22
KNC	74.08	73.54

En el MLPClassifier se consigue una mejora en ambos casos, respecto a la ejecución inicial, y volvemos a obtener la mejora comentada al eliminar los datos. Para el caso del KNC ambos cambios son un descenso del porcentaje del acierto.

Resultados obtenidos

Nayve-Bayes

- **Ventajas:** Es fácil y rápido predecir la clase de conjunto de datos de prueba. También funciona bien en la predicción multiclasa y se necesitan menos datos de entrenamiento.
- **Desventajas:** Si la variable categórica tiene una categoría en el conjunto de datos de prueba, que no se observó en el conjunto de datos de entrenamiento, el modelo asignará una probabilidad de 0 y no podrá hacer una predicción

Dentro de este algoritmo, he usado cuatro tipos:

- **Gaussian:** Supone que la probabilidad de las características es gaussiana.

```
gnb = GaussianNB()
modeloNBgau = gnb.fit(data_train, target_train)
predNBgau = modeloNBgau.predict(data_test)
scoresGau = cross_val_score(modeloNBgau, atributos, target, cv=5, scoring='accuracy')
```

- **Complement:** Es una adaptación del algoritmo estándar multinomial ingenuo de Bayes (MNB) que es especialmente adecuado para conjuntos de datos desequilibrados.

```
cnb = ComplementNB()
modeloNBcom = cnb.fit(data_train, target_train)
predNBcom = modeloNBcom.predict(data_test)
scoresCom = cross_val_score(modeloNBcom, atributos, target, cv=5, scoring='accuracy')
```

- **Bernoulli:** implementa los algoritmos ingenuos de clasificación y entrenamiento de Bayes para datos que se distribuyen de acuerdo con distribuciones de Bernoulli multivariadas.

```
bnb = BernoulliNB()
modelNBber = bnb.fit(data_train, target_train)
predNBber = modelNBber.predict(data_test)
scoresBer = cross_val_score(modelNBber, atributos, target, cv=5, scoring='accuracy')
```

- **Multinomial**: los datos se representan típicamente como conteos de vectores de palabras, aunque también se sabe que los vectores tf-idf funcionan bien en la práctica.

```
mnb = MultinomialNB()
modelNBMul = mnb.fit(data_train, target_train)
predNBMul = modelNBMul.predict(data_test)
scoresMul = cross_val_score(modelNBMul, atributos, target, cv=5, scoring='accuracy')
```

Análisis de resultados Nayve-Bayes:

Validación cruzada

Tipo	Acierto (%)
Gaussian	80.99
Complement	78.98
Bernoulli	77.20
Multinomial	78.86

Con esta tabla, podemos concluir que el tipo de Nayve-Bayes que más nos conviene utilizar será el Gaussian. Para comparar, el valor que usaré para la conclusión será el de 80.99 %

Matrices de confusión

Gaussian		Predicción	
		BENIGNO	MALIGNO
Clase Real	BENIGNO	43.17	10.14
	MALIGNO	9.88	36.80

Complement		Predicción	
		BENIGNO	MALIGNO
Clase Real	BENIGNO	41.87	11.44
	MALIGNO	10.36	36.54

Bernoulli		Predicción	
		BENIGNO	MALIGNO
Clase Real	BENIGNO	44.47	8.84
	MALIGNO	14.82	31.85

Multinomial		Predicción	
		BENIGNO	MALIGNO
Clase Real	BENIGNO	42.13	11.18
	MALIGNO	10.53	36.15

Árboles de decisión

- **Ventajas:** Fácil de entender e interpretar. Los árboles pueden ser visualizados. Requiere poca preparación de datos. El costo de usar el árbol es logarítmico en el número de puntos de datos usados para entrenar al árbol.
- **Desventajas:** Los alumnos del árbol de decisiones pueden crear árboles demasiado complejos que no generalizan bien los datos. Los participantes del árbol de decisión crean árboles sesgados si algunas clases dominan.

Dentro de este algoritmo, he usado dos tipos:

- **Árbol de decisión:** clase capaz de realizar una clasificación de varias clases en un conjunto de datos.

```
arbNor = tree.DecisionTreeClassifier()
arbNor = arbNor.fit(data_train, target_train)
predADnor = arbNor.predict(data_test)
scoresADnor = cross_val_score(arbNor, atributos, target, cv=5, scoring='accuracy')
```

- **Árbol de decisión extra:** Un clasificador de árboles extremadamente aleatorio

```
arbEx = tree.ExtraTreeClassifier()
arbEx = arbEx.fit(data_train, target_train)
predADex = arbEx.predict(data_test)
scoresADex = cross_val_score(arbEx, atributos, target, cv=5, scoring='accuracy')
```

Análisis de resultados Árboles de decisión:

Validación cruzada

Tipo	Acierto (%)
Normal	74.96
Extra	75.08

Con esta tabla, podemos concluir que el tipo de Árboles de decisión que más nos conviene utilizar será el Extra. El valor que usaré para la conclusión será el de 75.08 %

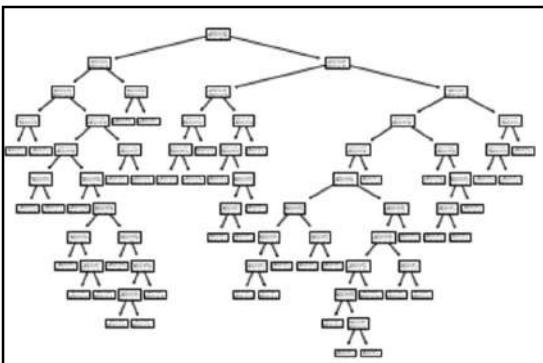
Matrices de confusión

		Predicción	
		BENIGNO	MALIGNO
Clase Real	BENIGNO	44.47	8.84
	MALIGNO	14.82	31.85

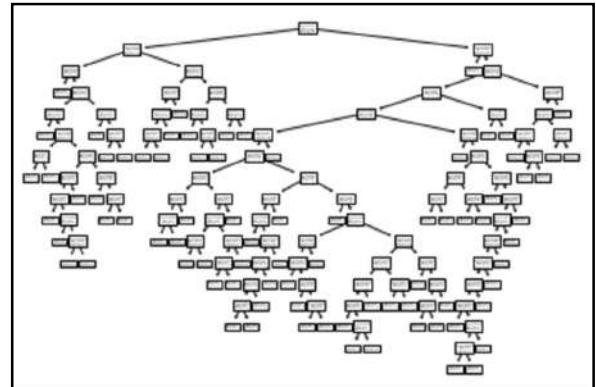
		Predicción	
		BENIGNO	MALIGNO
Clase Real	BENIGNO	37.97	15.34
	MALIGNO	8.45	38.23

El dibujo de los árboles, que ahora es casi imposible alcanzar a verlo, se verá mejorado cuando añadamos parámetros al análisis:

Normal:



Extra:



Support Vector Machines

- **Ventajas:** Eficaz en espacios de gran dimensión. Sigue siendo eficaz en los casos en que el número de dimensiones es mayor que el número de muestras.
- **Desventajas:** Si el número de características es mucho mayor que el número de muestras evitar el ajuste excesivo al elegir las funciones del núcleo y el término de regularización es crucial.

Dentro de este algoritmo he utilizado dos tipos:

- **NuSVC:** son métodos similares, pero aceptan conjuntos de parámetros ligeramente diferentes y tienen diferentes formulaciones matemáticas.

```
svr_nu = NuSVC()  
svr_nu.fit(data_train, target_train)  
predsvNu = svr_nu.predict(data_test)  
scoresNu = cross_val_score(svr_nu, atributos, target, cv=5, scoring='accuracy')
```

- **SVC:** el método de ajuste tomará como argumentos los vectores X y, en este caso particular, se espera que Y tenga valores de punto flotante en lugar de valores enteros.

```
svr_svc = SVC()  
svr_svc.fit(data_train, target_train)  
predsvSvc = svr_svc.predict(data_test)  
scoresSvc = cross_val_score(svr_svc, atributos, target, cv=5, scoring='accuracy')
```

Análisis de resultados SVM:

Validación cruzada

Tipo	Acierto (%)
NuSVC	80.52
SVC	78.15

Con esta tabla, podemos concluir que el tipo de SVM que más nos conviene utilizar será el NuSVC. El valor que usaré para la conclusión será el de 80.52 %

Matrices de confusión

NuSVC		Predicción	
		BENIGNO	MALIGNO
Clase Real	BENIGNO	43.56	9.75
	MALIGNO	10.66	36.02

SVC		Predicción	
		BENIGNO	MALIGNO
Clase Real	BENIGNO	42.65	10.66
	MALIGNO	14.43	32.24

Ensembled methods

- **Ventajas:** para cada ensayo que se agrupasen alrededor de un estimador central, y mostrarían tanta mayor dispersión alrededor de este valor cuanto más pequeño fuera su tamaño.
- **Desventajas:** se conceden un peso excesivo a los estudios con pequeño tamaño muestral

Dentro de este algoritmo he usado dos tipos:

- **Bagging meta-estimator:** toma como entrada un estimador base especificado por el usuario junto con parámetros, que especifican la estrategia para dibujar subconjuntos aleatorios.

```
bagging = BaggingClassifier()
bagging.fit(data_train, target_train)
preBag = bagging.predict(data_test)
scoresBag = cross_val_score(bagging, atributos, target, cv=5, scoring='accuracy')
```

- **Random Forests:** cada árbol en el conjunto se construye a partir de una muestra extraída con reemplazo (es decir, una muestra de arranque) del conjunto de entrenamiento.

```
forests = RandomForestClassifier()
forests.fit(data_train, target_train)
preFo = forests.predict(data_test)
scoresFo = cross_val_score(forests, atributos, target, cv=5, scoring='accuracy')
```

Análisis de resultados Ensembled Methods:

Validación cruzada

Tipo	Acierto (%)
Bagging	77.56
R. Forest	78.50

Con esta tabla, podemos concluir que el tipo de E. Methods que más nos conviene utilizar será el Beagging. El valor que usaré para la conclusión será el de 78.50 %

Matrices de confusión

Bagging		Predicción	
		BENIGNO	MALIGNO
Clase Real	BENIGNO	43.56	9.75
	MALIGNO	13.13	33.55

R. Forests		Predicción	
		BENIGNO	MALIGNO
Clase Real	BENIGNO	44.86	8.45
	MALIGNO	14.69	31.98

Redes neuronales

- **Ventajas:** Es fácil y rápido predecir la clase de conjunto de datos de prueba. También funciona bien en la predicción multiclasa y se necesitan menos datos de entrenamiento.
- **Desventajas:** Si la variable categórica tiene una categoría en el conjunto de datos de prueba, que no se observó en el conjunto de datos de entrenamiento, el modelo asignará una probabilidad de 0 y no podrá hacer una predicción.

Dentro de este algoritmo he usado dos tipos:

- MLPClassifier: algoritmo de perceptrón multicapa (MLP) que se entrena mediante la retro propagación.

```
modelMLP = MLPClassifier()
modelMLP.fit(data_test, target_test)
scores = cross_val_score(modelMLP, atributos, target, cv=5, scoring='accuracy')
salida=modelMLP.predict(data_test)
```

- KNC: Clasificador que implementa el voto de los k vecinos más cercanos

```
KNC = KNeighborsClassifier()
KNC.fit(data_test,target_test)
salida2=KNC.predict(data_test)
score2 = cross_val_score(KNC, atributos, target, cv=5, scoring='accuracy')
```

Análisis de resultados Redes Neuronales:

Validación cruzada

Tipo	Acierto (%)
MLPClassifier	80.04
KNC	80.03

Con esta tabla, podemos concluir que el tipo de redes neuronales que más nos conviene utilizar será el MLPClassifier. El valor que usaré para la conclusión será el de 80.04 %

Matrices de confusión

		Predicción	
		BENIGNO	MALIGNO
Clase Real	BENIGNO	42.26	11.05
	MALIGNO	7.41	39.27

		Predicción	
		BENIGNO	MALIGNO
Clase Real	BENIGNO	52.05	0.26
	MALIGNO	13.26	33.42

Análisis de resultados

Caso 1: Eliminando valores perdidos y sin parámetros

Con los porcentajes de acierto, de los tipos de algoritmos que mejor resultado nos han dado, obtenemos esta tabla. Para realizar esta valoración, se han eliminado todas las filas que tenían un valor perdido, por lo que los casos con los que se ha realizado el estudio han sido mucho más pequeños y se han usado los algoritmos por defecto.

Algoritmo	Porcentaje (%)
Nayve-Bayes	80.99
Árboles de decisión	75.08
S. Vector M.	80.52
Ensembled Methods	78.50
Redes Neuronales	80.04

Con los datos conseguidos, podemos determinar que el algoritmo con el que obtenemos mejor tasa de acierto, en estas condiciones, será el algoritmo de Nayve-Bayes y el peor será el de Árboles de decisión.

Caso 2: Modificando valores y sin parámetros

Obteniendo los porcentajes de acierto, de los tipos de algoritmos que mejor resultado nos han dado, conseguimos esta tabla. Para realizar esta valoración, se han sustituido los valores perdidos por el valor que más frecuencia tiene (el que más se repite), por lo que los casos con los que se ha realizado el estudio han sido los que nos dan desde el inicio. El inconveniente que tiene es que los datos van a ser parseados no se corresponden con un análisis correcto, es decir, vamos a poner un valor a la fuerza sin saber si es correcto. Volvemos a usar los algoritmos por defecto.

Algoritmo	Porcentaje (%)
Nayve-Bayes	81.16
Árboles de decisión	77.83
S. Vector M.	81.27
Ensembled Methods	80.54
Redes Neuronales	79.81

Algoritmo	Porcentaje (%)
Nayve-Bayes	
Gaussian	81.16
Complement	78.77
Bernoulli	77.63
Multinomial	78.98
Árboles de decisión	
Normal	77.72
Extra	77.83
S. Vector M.	
NuSVC	81.27
OneClass	22.16
SVM	79.60
Ensembled Methods	
Bagging	78.98
R. Forest	80.54
Redes Neuronales	
MLPClassifier	79.81
KNC	79.70

Mostramos en una tabla, el porcentaje más alto de cada tipo de algoritmo.

Comparativa de resultados:

Algoritmo	Inicial (%)	Primera Mejora (%)
Nayve-Bayes	80.99	81.16
Árboles de decisión	75.08	77.83
S. Vector M.	80.52	81.27
Ensembled Methods	78.50	80.54
Redes Neuronales	80.04	79.81

Tras comparar los porcentajes obtenidos, nos damos cuenta como hay algoritmos que han mejorado al tener más datos con los que trabajar. Lo distinguimos porque ahora el algoritmo que más porcentaje tiene es el de SVM con un 81,27 % y el peor, aunque han mejorado en un punto, sigue siendo los árboles de decisión.

Estos datos, desde mi punto de vista, no son muy fiables. Es verdad, que los porcentajes superan el 77,5 % pero los datos con los que han trabajado los algoritmos han sido valores forzados en algunos casos. En todas las filas que tenían una interrogación se le ha puesto el valor de más frecuencia a la ‘fuerza’. Por tanto, no se estarían respetando de forma adecuada los valores obtenidos en un supuesto hospital pudiendo afectar al análisis.

Nosotros imponemos que un determinado análisis tenga un valor forzado. Por tanto, para luego analizarlo donde hemos puesto un valor Y, debe estar puesto un valor F y, en consecuencia, se da un falso negativo o positivo que afecta al acierto del algoritmo.

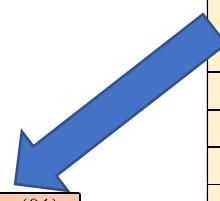
Caso 3: Modificando y eliminando valores y con parámetros

Ahora mostraremos el estudio realizado con la mejora en los algoritmos, añadiéndoles parámetros y datos.

Con parámetros y modificando los datos

Para el algoritmo de Nayve-Bayes, utilizaremos el porcentaje inicial debido a que no podemos añadir parámetros.

Algoritmo	Porcentaje (%)
Nayve-Bayes	81.16
Árboles de decisión	81.68
S. Vector M.	81.27
Ensembled Methods	83.14
Redes Neuronales	81.06



Algoritmo	Porcentaje (%)
Nayve-Bayes	
Gaussian	81.16
Complement	78.77
Bernoulli	77.63
Multinomial	78.98
Árboles de decisión	
Normal	81.68
Extra	76.38
S. Vector M.	
NuSVC	81.27
OneClass	22.16
SVM	79.60
Ensembled Methods	
Bagging	83.14
R. Forest	79.81
Redes Neuronales	
MLPClassifier	81.06
KNC	74.08

A la luz de estos resultados, el mejor algoritmo será el de Ensembled methods con un 83,14 %, siendo esta la mejor tasa obtenida. El peor algoritmo será el support vector machines con un 81,06 %, siendo este el que mejor resultado nos dio anteriormente. Ahora observamos, como la media ha subido hasta un 81%.

Realizamos la triple comparativa:

Algoritmo	Inicial (%)	Primera Mejora (%)	Segunda Mejora (%)
Nayve-Bayes	80.99	80.54	81.16
Árboles de decisión	75.08	76.58	81.68
S. Vector M.	80.52	80.95	81.27
Ensembled Methods	78.50	80.33	83.14
Redes Neuronales	80.04	79.60	81.06

A simple vista, observamos como todos los algoritmos han mejorado respecto al análisis inicial, algo que no había ocurrido con la primera mejora. Por tanto, podemos decir que, usando parámetros, se obtiene una mejora.

Con parámetros y eliminando los valores perdidos

Para el algoritmo de Nayve-Bayes, utilizaremos el porcentaje inicial ya que no se pueden añadir parámetros.

Algoritmo	Porcentaje (%)
Nayve-Bayes	80.99
Árboles de decisión	82.04
S. Vector M.	80.52
Ensembled Methods	83.00
Redes Neuronales	81.22



Algoritmo	Porcentaje (%)
Nayve-Bayes	80.99
Gaussian	78.98
Complement	77.20
Bernoulli	78.86
Multinomial	
Árboles de decisión	
Normal	82.04
Extra	77.21
S. Vector M.	
NuSVC	80.52
OneClass	22.43
SVM	78.15
Ensembled Methods	
Bagging	83.00
R. Forest	78.03
Redes Neuronales	
MLPClassifier	81.22
KNC	73.54

Realizamos la comparativa final:

Algoritmo	Inicial (%)	Primera Mejora (%)	Segunda Mejora (%)	Tercera mejora (%)
Nayve-Bayes	80.99	80.54	81.16	80.99
Árboles de decisión	75.08	76.58	81.68	82.04
S. Vector M.	80.52	80.95	81.27	80.52
Ensembled Methods	78.50	80.33	83.14	83.00
Redes Neuronales	80.04	79.60	81.06	81.22

Queda reflejado que reprocesar los datos nos da una mejora muy importante. En algunos casos, hemos comprobado como eliminar los datos perjudica en la tasa de acierto, pero procesarlos también nos afecta. La mejor tasa la obtenemos con el algoritmo de SVM con un 83,14 % cuando hay procesamiento de los datos y se añaden parámetros. Se ha aumentado la media un 3 % más.

Podemos concluir diciendo que cuando tenemos muchos datos, los árboles de decisión son muy útiles, y para otro tipo, podemos elegir entre varios algoritmos.

¿Hay algoritmos mejores?

Para realizar ese análisis, vamos a comparar los algoritmos intentando ver por qué es mejor un algoritmo en unos casos y en otros no.

La primera medida que vamos a comprobar es la predicción errónea, también conocida como f1_score. En este problema, teniendo en cuenta que manejamos datos sobre cáncer es mejor un falso positivo, es decir, predecir que es maligno, y que finalmente sea benigno, a un falso negativo, que sea maligno y se prediga benigno porque, en este último, la vida del paciente corre mucho peligro. Para ello, vamos a comparar la tasa de acierto con la diferencia de porcentajes entre los falsos positivos y falsos negativos. En mi caso, cuanto mayor sea el número, más falsos positivos habrá detectado y será “más” favorable. En caso de que el número sea negativo, se habrán detectado más falsos negativos y este algoritmo pondría en peligro a los pacientes.

Con Naive-Bayes, basado en el Teorema de Bayes, a pesar de ser uno de los algoritmos más simples y poderosos para la clasificación con una suposición de independencia entre los predictores, asume que el efecto de una característica particular en una clase es independiente de otras características. Esta suposición se denomina independencia condicional de clase. Por tanto, este algoritmo depende de sus predictores y, podemos decir que depende de cuanta importancia le da a las clases. Al igual que los árboles de decisión, que son representaciones gráficas de posibles soluciones a una decisión basadas en ciertas condiciones, es uno de los algoritmos de aprendizaje supervisado más utilizados en machine learning y pueden realizar tareas de clasificación o regresión. La comprensión de su funcionamiento suele ser simple y a la vez es muy potente. Las combinaciones para decidir el mejor árbol serían cientos o miles... Esto ya no es un trabajo para hacer manualmente y ahí es donde este algoritmo cobra importancia, pues él nos devolverá el árbol óptimo para la toma de decisión más acertada desde un punto de vista probabilístico. Por tanto, según las decisiones que tome el algoritmo tendremos unos resultados u otros.

Se recomienda el uso de algoritmos de SVM para desarrollar mejores modelos que permitan mayor precisión para la predicción del riesgo. Después de analizar el comportamiento de los distintos algoritmos, se aprecia que random forest (RF) representa una mejor opción al momento de tener que seleccionar un algoritmo para una clasificación binaria de 20.136 objetos. El algoritmo random forest presenta un resultado similar al de las redes neuronales, sin embargo, es importante entender que el algoritmo es completamente aleatorio. Por esta razón, mejorar los resultados de este algoritmo realizando un cambio en sus parámetros no garantiza obtener mejores resultados, debido a que, para cada ejecución, obtendríamos un valor diferente, que podría ser la mejor tasa de acierto, o la peor.

Por tanto, como ya se ha dicho anteriormente, no hay algoritmos mejores que otros, sino que algunos funcionarán mejor para unos casos y otros para otros. Seremos nosotros quien, según las características del problema, optaremos por usar unos algoritmos u otros.

Nayve-Bayes

Vemos como a pesar de que el algoritmo Gaussian tiene una mayor tasa de acierto, predice prácticamente la misma cantidad de FP como de FN. Por otra parte, se hace evidente que el peor da también un resultado negativo.

Algoritmo	Tasa de acierto (%)	FP - FN (%)
Gaussian	81.16	0.26
Complement	78.77	1.30
Bernoulli	77.63	-5.98
Multinomial	78.98	0.65

Árboles de decisión

El algoritmo que mejor tasa de acierto tiene, nos da una diferencia negativa y es perjudicial. Sin embargo, el extra nos da una diferencia positiva muy grande.

Algoritmo	Tasa de acierto (%)	FP - FN (%)
AD Normal	82.04	-5.98
AD Extra	77.21	6.89

Support Vector Machine

En estos algoritmos obtenemos unos resultados negativos en los dos casos. Otra vez, aunque se obtiene una tasa de acierto alta, también su predicción de falsos negativos es superior a la de positivos, algo que perjudica en el análisis.

Algoritmo	Tasa de acierto (%)	FP - FN (%)
NuSVC	81.27	-0.91
SVC	79.60	-3.77

Ensembled Methods

En este apartado, nos llevamos la gran sorpresa. La mejor tasa de acierto analizada en cualquiera de los algoritmos anteriormente citados, tiene más falsos negativos. Lo que nos puede reflejar que, aunque la tasa de acierto es muy buena, podría tener problemas al ser usado en algún programa médico.

Algoritmo	Tasa de acierto (%)	FP - FN (%)
Bagging	83.14	-3.38
R. Forest	79.81	-6.24

Redes neuronales

Aquí vemos dos cosas reflejadas. En el primer algoritmo, que ya fue el primero en una de las ejecuciones, también predice de una forma más correcta para un análisis médico. Por el contrario, el algoritmo que tiene una tasa de acierto baja, también realiza una mala predicción.

Algoritmo	Tasa de acierto (%)	FP - FN (%)
RN MLPClassifier	81.06	3.64
RN KNC	74.08	-13.03

Interpretación general

Para concluir, realizaremos un nuevo análisis con los últimos datos obtenidos basándonos en los dos parámetros. ¿Cuál sería el mejor algoritmo para ser utilizado en una aplicación de análisis médico?

Algoritmo	Tasa de acierto (%)	FP - FN (%)	Cociente (T.A / FP-FN)
Nayve-Bayes	81.16	0.26	312.15
Árboles de decisión	82.04	-5.98	-13.71
S. Vector M.	81.27	-0.91	-89.30
Ensembled Methods	83.14	-3.38	-24.59
Redes Neuronales	81.06	3.64	22.26

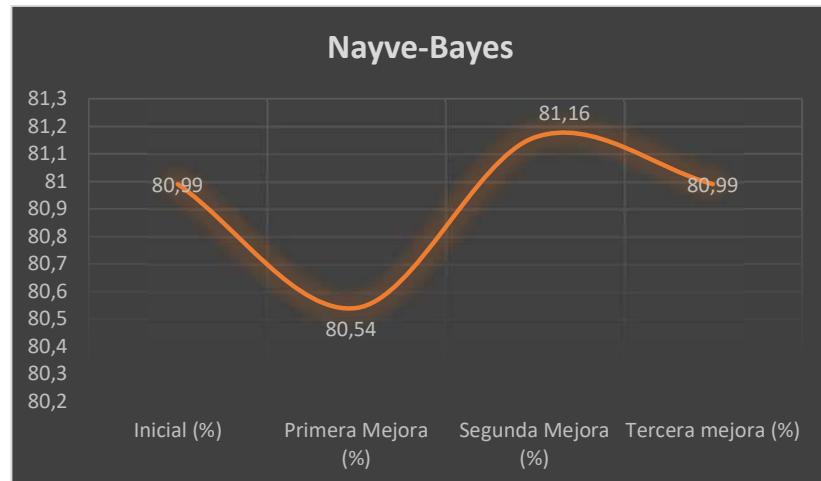
Pensando en un límite matemático, si se divide un número entre otro que es muy cercano a 0, este número tiende a infinito. Supongamos que todos los algoritmos dan más falsos positivos que negativos (en la tercera columna todos los valores son positivos) y que el mejor valor en esa columna es 0 (no hay ni falsos negativos ni falsos positivos). Vemos como en el cociente el mejor algoritmo en esa relación sería el de Nayve-Bayes Gaussiano. Aunque no tenga la mejor tasa de aciertos, vemos que se equivoca de forma “positiva” (realiza más falsos positivos que negativos). No obstante, puede ser que produzca el mismo número de errores (falsos positivos y falsos negativos), lo que no sería tampoco muy recomendable. Que un algoritmo tenga mucha diferencia positiva en la tercera columna tampoco es muy beneficioso, ya que significa que falla en la predicción, aunque el resultado sea favorable médicaamente hablando.

De esta forma, podemos concluir que no hay un algoritmo mejor que otro, sino que debido a las características del problema y los datos que tengamos, nos beneficiarán mejor unos que otros. Debemos tener en cuenta, qué queremos primar, si una tasa alta de acierto, aunque pueda cometer errores de fatal desenlace para un paciente (Support Vector Machines), o si queremos un algoritmo que pueda fallar, pero no sea perjudicial para el paciente (Redes neuronales), aunque tenga un acierto menor, o si queremos que se ajuste lo máximo a sólo predecir (Nayve-Bayes).

Interpretación de resultados

Nayve-Bayes

Con este algoritmo, tenemos el problema de que no permite parámetros. La conclusión que podemos realizar es que cuanto menos datos tenga, mayor es su tasa de acierto, aunque la diferencia es ínfima.



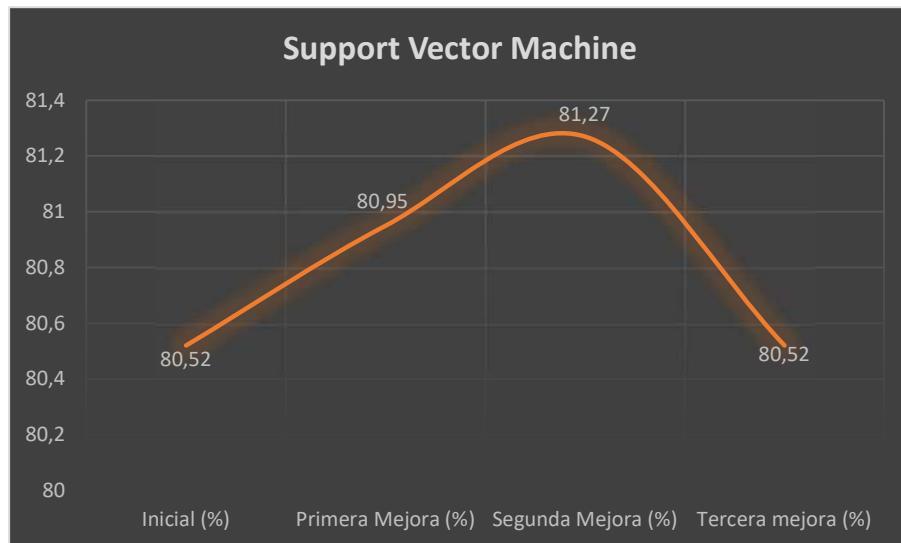
Árboles de decisión

Cuando no tiene parámetros este algoritmo su tasa de acierto es muy baja, sin embargo, con muchos valores y con parámetros, es uno de los mejores algoritmos que hemos usado. Su porcentaje supera el 81,5 % de acierto.



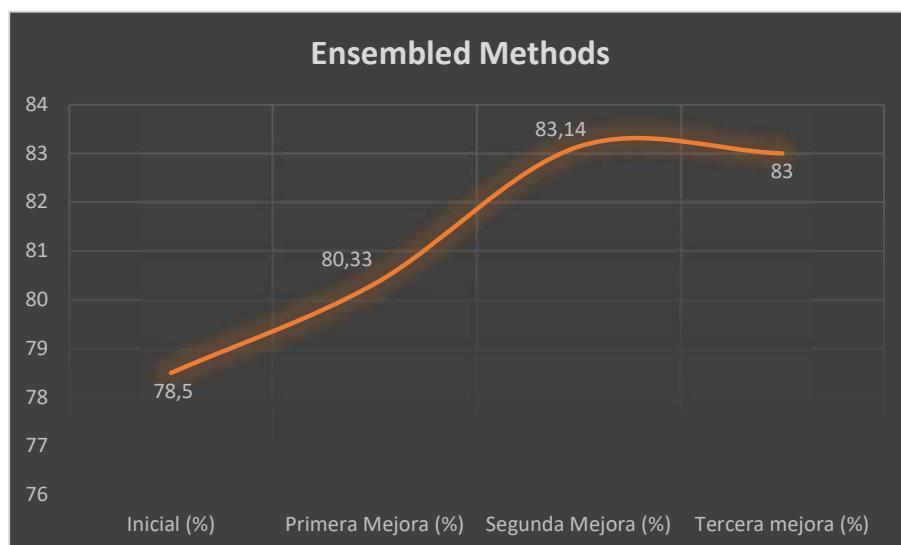
Support Vector Machine

Aunque ha tenido buena media y ha sido el mejor en la segunda mejora, hemos obtenido siempre los mismos valores. Podemos decir que no hay mejoras aceptables para este tipo de algoritmo. Muy similar al algoritmo de Nayve-Bayes.



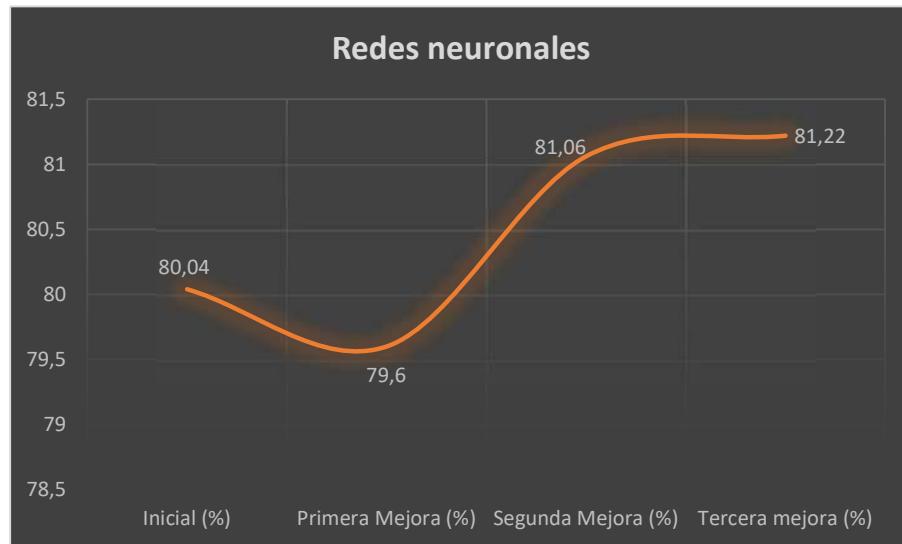
Ensembled methods

Este algoritmo ha comenzado dando unos resultados no muy buenos, sin embargo, con la última mejora, se ha convertido en el mejor algoritmo para resolver el problema de clasificación y aporta más claridad a la hipótesis planteada por mí. El resultado de tasa de acierto del 83,14 % es el único que ha superado la barrera del 82 %.



Redes neuronales

Aunque su media es superior al 80 %, en todos los casos se ha quedado en cuarta o tercera posición. Concluimos que este algoritmo no es muy útil para el problema planteado, por lo que no sería uno de los que se pudieran elegir para implementar un programa médico.



Contenido adicional

Para cada algoritmo he usado varios tipos, porque necesito varios porcentajes para obtener un valor correcto para realizar el estudio, ¿por qué? Si en mi caso, en vez de usar varios tipos hubiera usado uno sólo, por ejemplo en la segunda medición, hubiera podido ocurrir esto:

Algoritmo	Porcentaje (%)
Nayve-Bayes	
Gaussian	80.54
Complement	78.25
Bernoulli	77.63
Multinomial	78.14
Árboles de decisión	
Normal	76.58
Extra	75.65
S. Vector M.	
NuSVC	80.95
SVM	78.87
Ensembled Methods	
Bagging	77.93
R. Forest	80.33
Redes Neuronales	
MLPClassifier	79.39
KNC	79.60

Como hemos comentado anteriormente, el mejor algoritmo en la segunda medición es el SVM, gracias a que su tipo NuSVC nos da un 80,95 %, pero si en vez de usar ese tipo, hubiese escogido el SVM, me hubiese dado un 78,87 % y por ende, el mejor algoritmo, mantenido el resto, hubiese sido el Nayve-Bayes, dando un estudio incorrecto.

Para mi práctica voy a realizar cuatro mediciones.

La primera será una ejecución por defecto y eliminando los valores perdidos. De esta forma, habrá menos filas.

En la segunda medición, seguiré usando los algoritmos por defecto, pero esta vez, en vez de eliminar los valores perdidos, los cambiaré por el valor más frecuente.

Este es el motivo por el que hago la cuarta medición. Como ya he explicado anteriormente, quiero investigar sobre qué es mejor, bien eliminar datos perdidos y, por tanto, tener menos datos, o bien modificar manualmente unos datos, e intentar predecir sobre unos datos no reales. Ha quedado reflejado que es mejor mejorar los algoritmos y eliminar los datos.

En la tercera medición, modifco los datos y añado algunos parámetros.

En la cuarta medición, elimino los valores y añado algunos parámetros.

Bibliografía

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html>

<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

<https://scikit-learn.org/stable/modules/neighbors.html>

https://scikit-learn.org/stable/modules/gaussian_process.html

<https://scikit-learn.org/stable/modules/tree.html>

https://scikit-learn.org/stable/modules/neural_networks_supervised.html

<https://scikit-learn.org/stable/modules/multiclass.html>