Escuela Politécnica Superior

UNIVERSIDAD AUTONOMA DE MADRID

Fundamentos de Aprendizaje Automático 2019/2020

PRÁCTICA Nº 1

1. Objetivo

Esta práctica tiene varios objetivos. El primero consiste en analizar e implementar dos estrategias de particionamiento del conjunto de datos al que se aplica el aprendizaje: se trata de los métodos de *validación cruzada* y retención o *validación simple* [1]. Estas estrategias permitirán obtener los conjuntos de entrenamiento y prueba para todos los algoritmos desarrollados en las prácticas. El segundo objetivo de la práctica es implementar el algoritmo de clasificación *Naive-Bayes*. El algoritmo *Naive-Bayes* clasifica cada instancia/ejemplo en la clase cuya probabilidad a posteriori es máxima, suponiendo que todos los atributos con los que se representan las instancias son independientes. Finalmente, llevaremos a cabo una evaluación de las hipótesis obtenidas con este clasificador mediante el Análisis ROC.

2. Preliminares

TAREA

La clasificación es una de las principales tareas en el campo de la minería de datos. Cada instancia o ejemplo pertenece a una clase que se representa mediante un atributo de clase. El resto de los atributos de la instancia se utilizan para predecir la clase.

EVALUACIÓN

La clasificación permite construir modelos a partir de un conjunto de datos. El modelo puede ayudar a resolver un problema pero es necesario evaluar su calidad antes de utilizarlo en un entorno real. Para ello es necesario un proceso de evaluación que nos ofrezca alguna medida objetiva de la precisión del modelo creado. Para evaluar un modelo se podría utilizar el propio conjunto de entrenamiento como referencia, pero esta aproximación conduce normalmente a lo que se conoce como *sobreajuste*, es decir, la tendencia del modelo a trabajar bien con conjuntos de datos similares al de entrenamiento pero poco generalizable a otros datos. Lo más habitual por tanto es utilizar conjuntos diferentes para crear el modelo (*datos de entrenamiento*) y para evaluarlo (*datos de prueba o validación*).

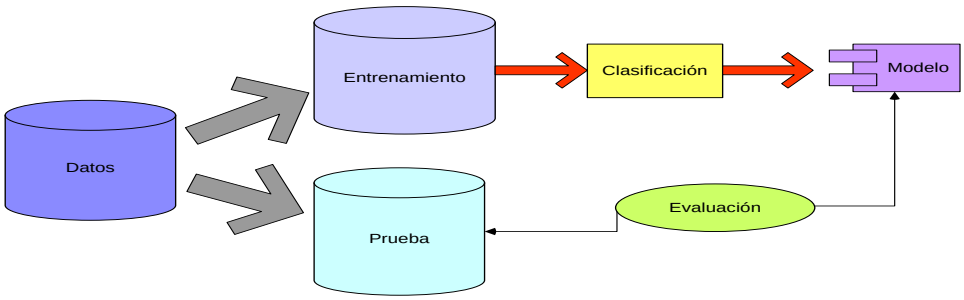

Figura 1: Evaluación de un modelo mediante conjuntos de entrenamiento y prueba [1]

La utilización de dos conjuntos de datos distintos puede ayudar a resolver problemas como el sobreajuste, pero no está exenta de otro tipo de problemas, como la disponibilidad de pocos datos o la dependencia del modo en que se realice la partición de los datos.

ESTRATEGIAS DE PARTICIONADO

Una manera de evitar los problemas de dependencia consiste en utilizar la técnica de *validación cruzada [1] (cross-validation)*. Con este método los datos se dividen en **k** grupos (comúnmente conocidos como *folds*), uno de los cuales se reserva para el conjunto de prueba y los **k-1** restantes se usan como conjunto de entrenamiento para construir el modelo. El modelo se evalúa entonces para predecir el resultado sobre los datos de prueba reservados. Este proceso se repite **k** veces, dejando cada vez un grupo diferente como conjunto de pruebas y el resto como conjunto de entrenamiento. De esta manera se obtienen **k** tasas de error, a partir de las que puede obtenerse la tasa de error promedio y la desviación típica de los errores, que nos darán la estimación de la precisión del modelo (ver Figura 2).

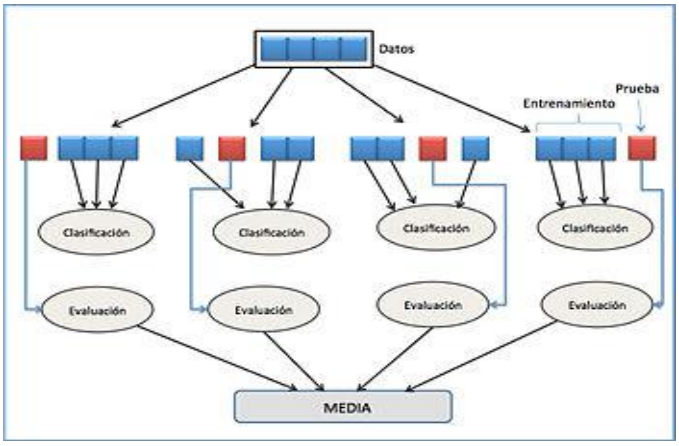

Figura 2: Estrategia de validación cruzada con cuatro grupos (folds). K=4. Fuente: https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada

Otra posibilidad es utilizar la estrategia de retención, *validación simple*, o hold-out donde se genera un solo conjunto de entrenamiento y un solo conjunto de prueba. Típicamente se especifica el tamaño deseado para una de estas dos particiones; por ejemplo, el tamaño del conjunto de entrenamiento (ver Figura 3). Cuando se usa esta técnica para evaluar métodos de aprendizaje automático, es común realizar varias particiones hold-out para diferentes permutaciones de los datos y, a partir de los errores obtenidos en cada permutación, calcular la media (y desviación típica).

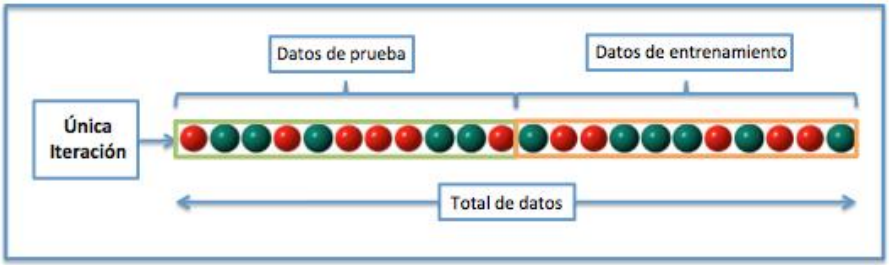

Figura 3: Ejemplo de estrategia hold-out con una sola partición de datos. Fuente: https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada

NOTA: Cuando se realiza el particionado es importante asegurarse de que las particiones son generadas sin ningún tipo de sesgo. Por ejemplo, si se tienen los patrones ordenados por clase y las particiones se generan sin permutar los datos, no se estaría obteniendo una muestra "realista" de la distribución subyacente en los datos.

3. Actividades

La planificación temporal sugerida y las actividades a llevar a cabo son las siguientes:

- *1º semana*: Implementar el diseño de la aplicación necesaria para organizar los clasificadores y crear las particiones para los conjuntos de entrenamiento y prueba de un clasificador. Se deben implementar los dos métodos de particionado explicados, validación cruzada y validación simple, pero el diseño debería permitir añadir cualquier otro tipo de particionado de forma flexible. Un posible diseño se comenta en el apartado 4. Probar los dos tipos de particionado con el conjunto de datos ***lenses*** y con el conjunto ***tic-tac-toe*** que ya empleamos en la práctica de iniciación.

- *2ª semana:* Implementar el clasificador *Naive-Bayes* y aplicar este algoritmo a los conjuntos de datos **lenses**, **tic-tac-toe** y **german**, para obtener los porcentajes de error y acierto en clasificación en cada caso, tanto con validación cruzada como con validación simple. En todos los casos se deben hacer los cálculos sin aplicar y aplicando la *corrección de Laplace*. Tened en cuenta que el conjunto de datos **german** incluye atributos continuos, por lo que en ese caso supondremos que estos atributos siguen una distribución normal dada la clase. Para ello se debe calcular la media y la varianza de cada atributo continuo. Pueden ser de utilidad las funciones para la distribución normal del módulo `scipy.stats.norm`.

- *3ª semana*: Comparar los resultados obtenidos con los que proporciona la librería de aprendizaje automático scikit-learn (http://scikit-learn.org/stable/). Reproducir el diseño experimental utilizado anteriormente para la validación simple y la validación cruzada. Algunos aspectos sobre el uso de scikit-learn se comentan en el apartado 5.

- *4ª semana*: Evaluar y comentar las hipótesis obtenidas con el clasificador Naive-Bayes. Los detalles se encuentran en el apartado 6.

4. Diseño

Con el objetivo de desarrollar una aplicación lo más flexible y general posible se sugiere un diseño basado en patrones cuyos detalles más importantes se muestran en la Figura 4 y se especifican a continuación. Este diseño es una sugerencia y puede ser modificado si se considera necesario. Las modificaciones realizadas (diseño de clases, nuevos atributos, etc.) deberán ser descritas en la entrega de la práctica. No obstante, se penalizarán diseños que incurran en problemas ya estudiados en asignaturas previas: redundancia, falta de adaptación a cambios, etc.

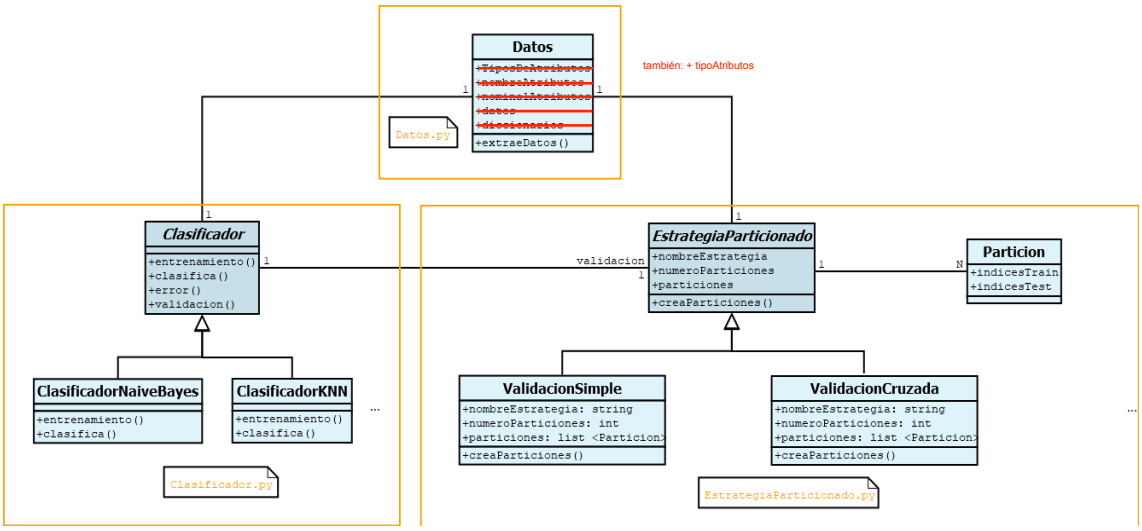

Figura 4: Esquema de clases y ficheros sugerido

PARTICIONADO

Para conseguir un particionado de los datos con la mayor flexibilidad posible se debe crear una clase abstracta *EstrategiaParticionado*, que separe la estrategia de particionado del particionado en sí mismo. Por su parte, la clase abstracta *Clasificador* utiliza una estrategia de particionado determinada para llevar a cabo su algoritmo de clasificación, empleando el conjunto de entrenamiento para construir el modelo y el conjunto de prueba para evaluarlo. Cada estrategia de particionado creará los conjuntos de entrenamiento y prueba de diferente forma según el algoritmo de particionado. En concreto, la *validación cruzada* dividirá los datos del fichero de entrada de forma aleatoria entre el número de particiones especificado. Por su parte, la *validación simple* dividirá los datos de entrada en dos conjuntos, uno para entrenamiento y otro para prueba en función del porcentaje que se especifique. Con el esquema propuesto es sencillo ir añadiendo diferentes estrategias de particionado en función de las necesidades de cada clasificador. La clase *Partición* contiene los conjuntos de índices para entrenamiento y prueba que se generan al utilizar cada tipo de particionado. Es importante tener en cuenta que estos índices son simplemente listas de números que nos permiten acceder a las filas correspondientes de la matriz datos que se calculó en la práctica de iniciación.

CLASIFICADORES

Como el objetivo de la asignatura es implementar y trabajar con diferentes clasificadores, la idea es crear una jerarquía de clasificadores, como se propone en la Figura 4. Los métodos entrenamiento y clasifica son auto-explicativos y cada uno utilizará el conjunto de índices de entrenamiento y prueba correspondiente, creados según el tipo de particionado. El método error calcula el número de aciertos y errores, comparando la clase predicha por el clasificador con el valor real de la clase, para cada instancia del conjunto de prueba. Finalmente, el método validación funciona como el método principal, calculando las particiones e iterando sobre el número de particiones calculadas para entrenar y clasificar sobre cada partición, obteniendo el error correspondiente a cada partición.

DATOS

En la clase Datos cuyo constructor se implementó en la práctica de iniciación, se implementará el método *extraeDatos* que recibirá una lista de índices correspondientes a un subconjunto de

patrones a seleccionar sobre el conjunto total de datos. El método devolverá la submatriz de datos que corresponde a los índices pedidos.

ESQUEMA GENERAL DE EJECUCIÓN

De acuerdo al diseño presentado, el entrenamiento y validación de un clasificador (`clasificador`) sobre un conjunto de datos (`dataset`) utilizando determinada estrategia de particionado (`estrategia`) podría realizarse del siguiente modo:

```
dataset=Datos('../ConjuntosDatos/tic-tac-toe.data')
estrategia=EstrategiaParticionado.ValidacionCruzada(…)
clasificador=Clasificador.ClasificadorNaiveBayes(…)
errores=clasificador.validacion(estrategia,dataset,clasificador)
```

Donde el array/lista `errores` contendría los errores obtenidos para cada uno de los datos de prueba de la estrategia correspondiente. Como puede verse, los constructores para *ValidaciónCruzada* o *ClasificadorNaiveBayes* no especifican los argumentos porque la decisión sobre lo que necesita cada clase se deja a vuestra elección, aunque como en todos los aspectos relativos al diseño, la elección de argumentos debe explicarse debidamente.

PLANTILLAS

En Moodle se encuentran los ficheros *Datos.py*, *Clasificador.py* y *EstrategiaParticionado.py* con las plantillas para las distintas clases presentadas en la Figura 4. El constructor de la clase Datos es el implementado en la práctica de iniciación.

5. Scikit-learn

Scikit-learn proporciona diferentes implementaciones para el algoritmo Naive Bayes (http://scikit-learn.org/stable/modules/naive_bayes.html). En esta práctica, consideraremos las siguientes funciones:

- **MultinomialNB**: se utiliza el conteo del número de veces que un atributo toma un determinado valor dada la clase. El valor a priori de cada clase se puede asumir uniforme (`fit_prior=False`) o se puede obtener a partir de los datos (`fit_prior=True`). Esta función también permite especificar un parámetro aditivo de suavizado `alpha`. `alpha=0` no realiza ningún tipo de suavizado y `alpha=1` implementa la corrección de Laplace.

- **`GaussianNB`**: la verosimilitud de las variables se asume que sigue una distribución normal.

Un aspecto importante a tener en cuenta cuando se utilizan funciones de la librería Scikit-learn es que muchos de los métodos no admiten atributos categóricos como tal y es necesario realizar un preprocesado de los datos con el fin de poder utilizar determinados algoritmos. Para ello, scikit-learn proporciona la clase **`OneHotEncoder`** (http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html) contenida en el submódulo `preprocessing`. Esta clase proporciona métodos para codificar valores enteros[1]

[1] Recordar que en el constructor de la clase Datos se habían codificado las variables categóricas como enteros

en bits de forma que si una variable toma **m** posibles valores enteros diferentes, ésta se codifica en m bits con solo uno de ellos activos. Por ejemplo, en el conjunto de datos *tic-tac-toe* donde cada una de las nueve variables toma tres posibles valores (`x,b,o`), tras la codificación *one-hot* (o *one-of-K*), el dataset pasa a tener 9x3=27 atributos.

De acuerdo al diseño sugerido para la clase `Datos`, es posible obtener esta codificación como sigue, asumiendo que la variable `dataset` es una instanciación de la clase `Datos`:

```
from sklearn import preprocessing

# Encode categorical integer features using a one-hot aka one-of-K
scheme (categorical features)
encAtributos =
preprocessing.OneHotEncoder(categorical_features=dataset.nominalAtribu
tos[:-1],sparse=False)
X = encAtributos.fit_transform(dataset.datos[:,:-1])
Y =dataset.datos[:,-1]
```

Por tanto, la variable `X` contendrá la matriz de atributos codificada y el array unidimensional `Y` contendrá la clase de cada patrón.

En cuanto al particionado de los datos, el submódulo **`model_selection`** de Scikit-learn implementa diferentes estrategias (http://scikit-learn.org/stable/model_selection.html). En lo que se refiere a las estrategias de validación simple y validación cruzada a implementar en esta práctica, estas son algunas de las funciones más relevantes:

- **Validación simple**: La función **`train_test_split`** hace transparente la gestión de índices ya que devuelve directamente los conjuntos de entrenamiento y validación para un conjunto de datos dado. Por su parte, la instanciación de la clase **`ShuffleSplit`** proporciona los índices para dividir los datos en subconjuntos de training y test. Los parámetros de estas funciones/constructores permiten indicar la proporción de patrones a utilizar en test o el número de particiones a generar, entre otros.

- **Validación cruzada**: Las funciones **`cross_val_score`** y **`cross_valpredict`** devuelven directamente los errores por validación cruzada por cada *fold* y las predicciones por cada patrón, respectivamente. Por su parte, la instanciación de la clase **`KFold`** genera los índices de las particiones. Entre otras posibilidades, los parámetros de estas funciones/constructores permiten indicar el número de *folds* a generar o imponer la permutación aleatoria de los datos antes de generar la partición.

6. Evaluación de hipótesis mediante Análisis ROC

En la implementación que hemos hecho en los apartados anteriores del clasificador Naive-Bayes hemos utilizado como medida de evaluación la precisión del mismo, es decir, el porcentaje de aciertos/errores en el conjunto de test. Esta medida, que se utiliza frecuentemente por ser muy simple, tiene el inconveniente de que asume que todos los errores tienen igual importancia, algo que generalmente no ocurre. Podemos encontrar medidas más apropiadas para evaluar los clasificadores y vamos a centrarnos en el caso de clasificadores binarios. Una técnica habitual en este caso es el *Análisis ROC* (Receiver Operating Characteristic).

En primer lugar calcularemos la **matriz de confusión** para los conjuntos de datos *lenses*, *tic-tac-toe* y *german*.

	REAL	
ESTIMADO	True Positives (TP)	False Positives (FP)
	False Negatives (FN)	True Negatives (TN)

A partir de los valores obtenemos las tasas correspondientes:

$$TPR = TP / (TP + FN)$$
$$FNR = FN / (TP + FN)$$
$$FPR = FP / (FP + TN)$$
$$TNR = TN / (FP + TN)$$

Dibuja el clasificador Naive-Bayes en el espacio ROC para cada conjunto de datos, sabiendo que en el eje Y representamos el valor TPR y en el eje X el valor FPR.

A la vista de la representación gráfica de los resultados, ¿qué conclusiones podemos extraer de la clasificación Naive-Bayes en cada uno de los conjuntos de datos?

7. Fecha de entrega y entregables

Semana del 28 de Octubre al 1 de Noviembre de 2019. La entrega debe realizarse antes del comienzo de la clase de prácticas correspondiente. Se deberá entregar un fichero comprimido .zip con nombre **`FAAP1_<grupo>_<pareja>.zip`** (ejemplo `FAAP1_1461_1.zip`) y el siguiente contenido:

1. **Ipython Notebook (.ipynb)**. El Notebook debe estructurarse para contener los siguientes apartados:

Apartado 1	**Particionado** Análisis de las dos estrategias de particionado propuestas: simple, y cruzada, para los conjuntos propuestos: lenses, german y tic-tac-toe. El análisis consiste en una descripción de los índices de train y test devueltos por cada uno de los métodos de particionado, junto con un comentario sobre las ventajas/desventajas de cada uno de ellos.
Apartado 2	**Naive-Bayes** Tabla con los resultados de la ejecución para los conjuntos de datos analizados (lenses, tic-tac-toe y german). Considerar los dos tipos de particionado. Los resultados se refieren a las tasas de error/acierto y deben incluirse tanto con la corrección de Laplace como sin ella. Se debe incluir tanto el promedio de error para las diferentes particiones como su desviación típica. Es importante mostrar todos los resultados agrupados en una

	tabla para facilitar su evaluación. Breve análisis de los resultados anteriores
Apartado 3	**Scikit-Learn** Incluir los mismos resultados que en el apartado 2 pero usando los métodos del paquete scikit-learn. Comparar y analizar los resultados.
Apartado 4	**Evaluación de hipótesis mediante Análisis ROC** Matriz de confusión y diagramas del clasificador en el espacio ROC

2. **Ipython Notebook exportado como html.**

3. Código Python (**ficheros.py**) necesario para la correcta ejecución del Notebook.

8. Puntuación de cada apartado

La valoración de cada apartado será la siguiente:

- ✓ **Particionado**: 2 puntos (0,5 punto para la validación simple y 1,5 puntos para la validación cruzada)
- ✓ **Naive-Bayes**: 4 puntos
- ✓ **Scikit-Learn**: 2 puntos
- ✓ **Análisis ROC**: 2 puntos

9. Referencias

[1] J. H. Orallo, M. J. Ramírez Quintana, C. F. Ramírez. *Introducción a la Minería de Datos*. Pearson, Prentice Hall