Fundamentos de Aprendizaje Automático 2016/2017

PRÁCTICA Nº 1

1. Objetivo

El objetivo de esta 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. Adicionalmente, se va a implementar de forma genérica el procedimiento de *validación cruzada* para poder evaluar los clasificadores desarrollados en las prácticas de la asignatura.

2. Preliminares

TAREA

La clasificación es una de las principales tareas en el campo de la minería de datos (*data mining*). 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, al igual que otros métodos de aprendizaje, 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 sin ser 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*).

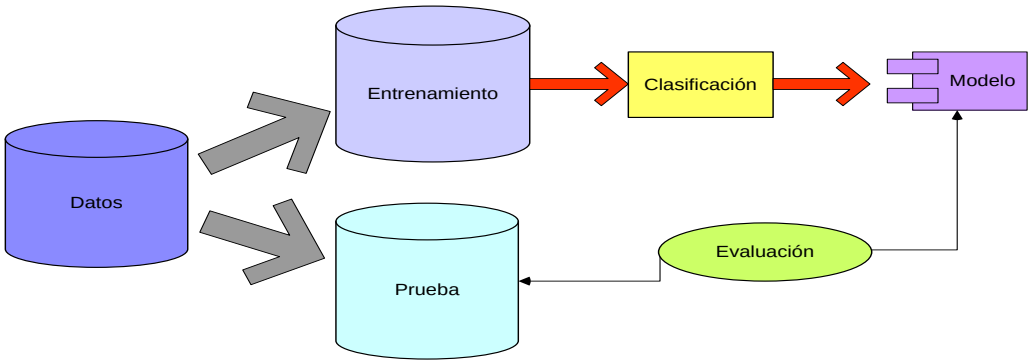

Figura 1 Evaluación de un modelo mediante conjuntos de entrenamiento y prueba

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.

Una manera de evitar los problemas de dependencia consiste en utilizar la técnica de *validación cruzada (cross-validation)*. Con este método los datos se dividen en **k** grupos (comúnmente conocidos como *folds*), uno de ellos 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).

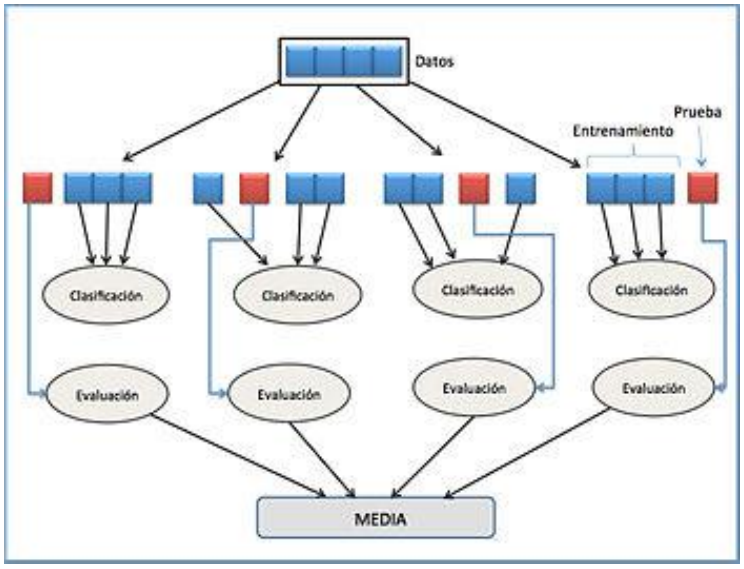

Figura 2 Estrategia de validación cruzada con cuatro grupos (folds). K=4. Fuente: https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada

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).

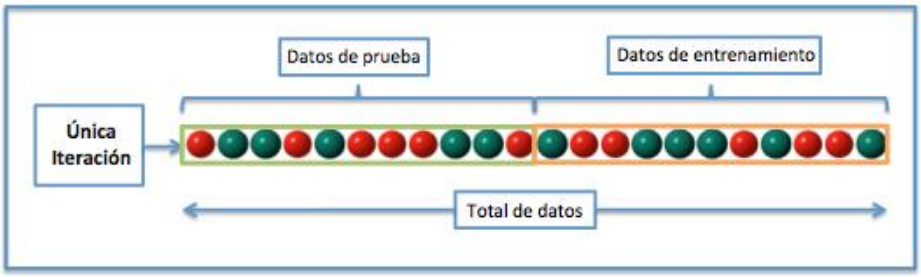

Figura 3 Ejemplo de estrategia hold-out con una sola partición de datos. Fuente: https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada

¡OJO! 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 la estructura 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 dos métodos de particionado distintos, validación cruzada y validación simple (un solo conjunto de entrenamiento y un solo conjunto de prueba), pero el diseño debería permitir añadir cualquier otro tipo de particionado de forma flexible. Una posible estructura de implementación se comenta en el apartado 4.

- *2º y 3ª semana:* Implementar el clasificador de *Naive-Bayes* y clasificar los patrones de los siguientes conjuntos de datos para obtener los porcentajes de error y acierto en clasificación para el conjunto de test, tanto con validación cruzada como con validación simple.

    - Conjunto de datos ***tic-tac-toe*** (http://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame)**.** Clasificar el mismo conjunto de datos aplicando la corrección de Laplace. Este conjunto ya se utilizó en la práctica 0.

    - Conjunto de datos ***wine*** (http://archive.ics.uci.edu/ml/datasets/Wine) para determinar el tipo de vino en base a propiedades químicas. Se proporciona junto con el enunciado de las prácticas el conjunto `wine_proc.data` con el formato de datos especificado en la práctica 0. Suponer que todos los atributos continuos siguen una distribución normal dada la clase. Para ello se debe calcular la media y la varianza de cada atributo continuo dada la clase. Puede ser de utilidad las funciones para la distribución normal del módulo `scipy.stats.norm`.

    - Conjunto de datos ***credit approval*** (http://archive.ics.uci.edu/ml/datasets/Credit+Approval). Suponer que todos los atributos continuos siguen una distribución normal dada la clase. Para ello se

debe calcular la media y la varianza de cada atributo continuo dada la clase. Este conjunto ya se utilizó en la práctica 0.

- Conjunto de datos de dígitos obtenido en las clases de teoría. Una versión procesada de los datos (`digits.data`) se proporciona con los conjuntos de datos de la práctica.

- *3º 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.

4. Lenguaje y Diseño

El lenguaje a utilizar para el desarrollo de la práctica será Python. 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. El esquema de clases debe ser respetado, aunque se pueden modificar/añadir atributos a las clases y parámetros a los métodos cuando se considere necesario. Las modificaciones deberán ser descritas y justificadas en la entrega de la práctica.

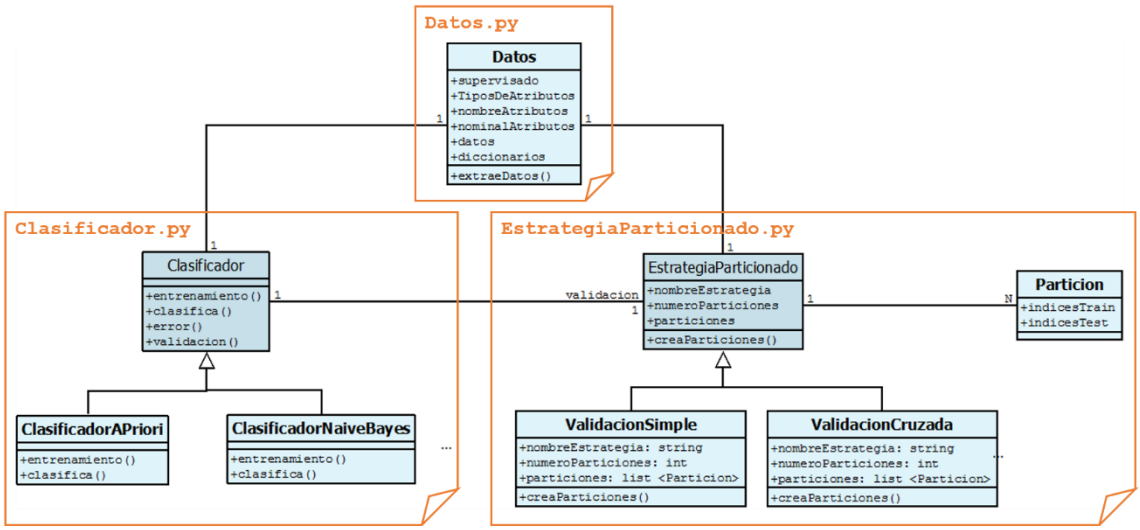

Figura 4 Esquema de clases y ficheros

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, tal y como se ha explicado anteriormente. 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. Se podrá especificar también un número determinado de particiones simples. Con el esquema propuesto es sencillo ir añadiendo diferentes estrategias de particionado en función de las necesidades de cada clasificador.

CLASIFICADORES

Como el objetivo de la asignatura es implementar y trabajar con diferentes clasificadores, la idea es crear una jerarquía de clasificadores, como se muestra en la Figura 4. Los métodos entrenamiento y clasifica son auto-explicativos y cada uno utilizará el conjunto de entrenamiento y prueba correspondiente, creados según el tipo de particionado.

DATOS

En la clase Datos cuyo constructor se implementó en la práctica 0, se implementará la función `extraeDatos` que recibirá una lista de índices correspondientes a un subconjunto de patrones a seleccionar sobre el conjunto total de datos.

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('../P0/ConjuntosDatos/tic-tac-toe.data',True)
estrategia=EstrategiaParticionado.ValidacionCruzada()
clasificador=Clasificador.ClasificadorAPriori()
errores=clasificador.validacion(estrategia,dataset,clasificador)
```

Donde el array/lista `errores` contendría los errores obtenidos para cada uno de los conjuntos de prueba de la estrategia correspondiente.

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 0.

La clase *ClasificadorAPriori* se puede usar para probar la validación cruzada. Como fichero de datos se puede usar el conjunto de datos ***tic-tac-toe***.

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] 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 **`cross_validation`** de Scikit-learn implementa diferentes estrategias (http://scikit-learn.org/stable/modules/cross_validation.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

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

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. Fecha de entrega y entregables

Semana del 17 al 21 de Octubre de 2016. 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)** con las instrucciones necesarias para realizar las pruebas descritas en el apartado 3 y el correspondiente análisis de resultados. El Notebook debe estructurarse para contener los siguientes apartados:

Apartado 1	Tabla con los resultados de la ejecución para los conjuntos de datos analizados (tic-tac-toe, wine, credit approval y dígitos). 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.
Apartado 2	Breve análisis de los resultados anteriores
Apartado 3	Para el conjunto **tic-tac-toe** mostrar los siguientes valores únicamente para la primera partición (validación cruzada): • ***Probabilidades a priori***: P(*Class=positive*) y P(*Class=negative*) • ***Probabilidades de máxima verosimilitud***: P(*MLeftSq=b|Class=positive*) P(*TRightSq=x|Class=negative*) • Las dos últimas probabilidades con la corrección de ***Laplace***
Apartado 4	Para el conjunto **credit approval** mostrar los siguientes valores únicamente para la primera partición (validación cruzada): • ***Probabilidades a priori***: P(*Class=+*) y P(*Class=-)* • ***Probabilidades de máxima verosimilitud***: P(*A7=bb|Class=+*) P(*A4=u|Class=-*) • ***Media*** y ***desviación típica*** de los atributos continuos *A2, A14* y *A15* condicionado al valor de clase +

Apartado 5	Incluir los mismos resultados que en el apartado 1 pero usando los métodos del paquete scikit-learn. Comparar y analizar los resultados.

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

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