Práctica obligatoria 1

Un sistema de representación mixto para detección de variedades

El siguiente enunciado corresponde a la primera práctica obligatoria de la asignatura Aprendizaje Automático 3, de 4º curso en el Grado en Inteligencia Artificial de la URJC.

1 Normas

La práctica se realizará de manera individual y se presentará usando el aula virtual.

La fecha límite de presentación es el martes 28 de octubre a las 23:00.

Para presentarla se deberá entregar un único fichero ZIP que contendrá el código fuente y un fichero PDF con la descripción del sistema desarrollado y las pruebas realizadas.

La puntuación de esta práctica corresponde al 20% de la asignatura. En particular se valorará:

- El correcto funcionamiento del sistema de software desarrollado haciendo uso de las técnicas explicadas en clase sobre los diferentes experimentos. (50%)
- La estructura y calidad del software realizado: clases, métodos, comentarios de los métodos y sus parámetros. (30%)
- La calidad del documento PDF. (20%)
- La modificación de dicha práctica que se solicitará el día 29 de octubre en horario de clase. Si dicha modificación no se realiza de manera totalmente correcta, la práctica estará suspensa a pesar de la nota que se derive de los puntos anteriores.

La práctica deberá ejecutarse sobre Python 3.12 y podrá apoyarse en `Numpy`, `Sklearn` y `Pytorch`.

Para ejecutar la práctica deberá escribirse en la consola de comandos "`python`" seguido del nombre un *script* en *python* ("`mixed_manifold_detector`"), del nombre de un fichero CSV para entrenar y opcionalmente de un segundo fichero CSV para test, sin ningún otro parámetro adicional. Al ejecutar este comando se mostrará por pantalla el entrenamiento y el resultado sobre la parte de test. Por ejemplo:

```
python mixed_manifold_detector.py mnist_train.csv mnist_test.csv
```

El código desarrollado en las prácticas debe ser original. La copia (total o parcial) de prácticas será sancionada, al menos, con el SUSPENSO global de la asignatura en la convocatoria correspondiente. En estos casos, además, no regirá la liberación de partes de la asignatura (habrá que volver a presentarse al examen) y podrá significar, en la siguiente convocatoria y a discreción del profesor, el

tener que resolver nuevas pruebas y la defensa de las mismas de forma oral. Las sanciones derivadas de la copia, afectarán tanto al alumno que copia como al alumno copiado.

Para evitar que cuando se usa código de terceros sea considerado una copia, se debe citar siempre la procedencia de cada parte de código no desarrollada por el propio alumno (con comentarios en el propio código y con mención expresa en la memoria de las referencias). El plagio o copia de terceros (por ejemplo, de una página web) ya sea en el código a desarrollar en las prácticas y/o de parte de la memoria de las prácticas, sin la cita correspondiente, acarrearán las mismas sanciones que en la copia de prácticas de otros alumnos.

2 Introducción

Se desea construir una clase llamada `MixedManifoldDetector` en Python que permita la representación en 2D de los patrones de cualquier problema de aprendizaje automático no supervisado.

Para ello, esta clase utiliza un enfoque mixto (ver Figura 1). Primero, entrena un *autoencoder* con los datos de entrenamiento proporcionados. Después, entrena un sistema de *manifold* sobre la representación latente creada por el *autoencoder.*

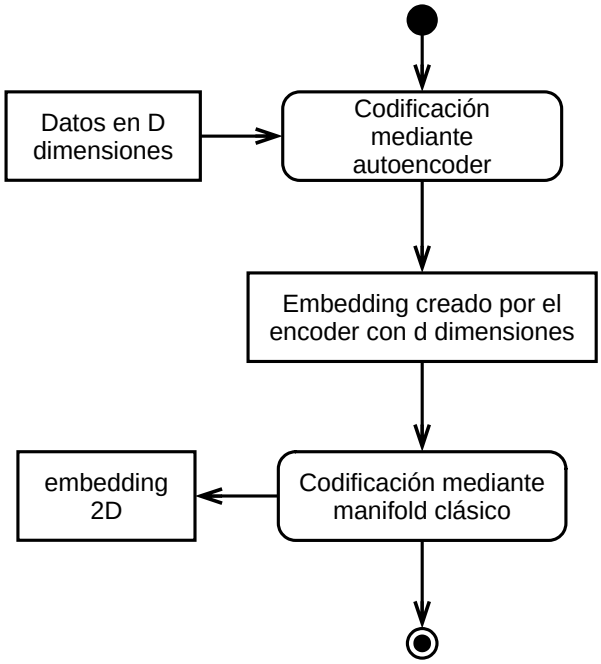

Figura 1: Diagrama de actividad del sistema deseado.

El sistema que se desarrolle debe venir acompañado de un fichero en python llamado `mixed_manifold_detector.py` . Dicho fichero contendrá una función principal que permita su ejecución utilizando el siguiente comando:

```
python mixed_manifold_detector.py nombre_fichero_csv
```

El fichero CSV que admitirá el sistema consistirá en un fichero de texto que contiene valores numéricos reales separados por comas. Cada fila corresponderá a un patrón. Cada columna corresponderá a una característica. La primera columna corresponderá a la etiqueta del patrón. La primera fila corresponderá a los nombres de las características. La Figura 2 presenta un ejemplo de un fichero CSV con las características descritas.

label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11
0	0	0	0	0	0	0	0	9	8	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	14	53	99	17	0
2	0	0	0	0	0	0	0	0	0	161	212

Figura 2: Ejemplo de fichero CSV que el sistema deberá admitir.

3 La clase MixedManifoldDetector

La clase `MixedManifoldDetector` deberá tener al menos los siguientes métodos:

- `__init__(self, autoencoder :Autoencoder, manifold_alg: sklearn.base.TransformerMixin)`
- `fit_transform(self, data : np.ndarray) : np.ndarray`
- `fit(self, train_data : np.ndarray)`
- `transform(self, data : np.ndarray) : np.ndarray`

3.1 El constructor

Esta clase inicializa los objetos de `MixedManifoldDetector` con los dos elementos principales que usará durante su operativa: el *autoencoder* y la técnica clásica de *manifold*.

El constructor de esta clase recibe dos parámetros obligatorios:

- `autoencoder : Autoencoder` – Un objeto de la clase `Autoencoder` que contendrá el modelo en `pytorch` del *autoencoder* que se utilizará como primera parte del sistema. La clase `Autoencoder` se explica posteriormente. Si no se especifica un *autoencoder*, `MixedManifoldDetector` pondrá por defecto un modelo con 3 capas lineales en el *encoder*, una capa de 32 neuronas para el *embedding* y 3 capas lineales en el *decoder*.
- `manifold_alg: sklearn.base.TransformerMixin` – Un objeto de la clase `TransformerMixin` que corresponderá al algoritmo de *manifold* que utilizará como segunda parte del sistema. Por defecto, si no se especifica otro modelo, `MixedManifoldDetector` utilizará `TSNE`.

3.2 El método fit_transform

Este método recibirá una matriz de `numpy` con los patrones de entrenamiento. Cada fila corresponderá a un patrón y cada columna a una característica discriminante.

A continuación, realizará el entrenamiento del *autoencoder* utilizando dichos patrones.

Finalmente, devolverá una matriz de `numpy` con las coordenadas 2D asociadas a cada patrón proporcionado en el entrenamiento. Así, en dicha matriz de salida, la fila 0 corresponderá a las coordenadas (x,y) del patrón 0, la fila 1 corresponderá a las coordenadas (x,y) del patrón 1…

El método `fit` simplemente realizará una llamada a `fit_transform` y no devolverá nada.

3.3 El método transform

Este método recibe:

- Una matriz de `numpy` con datos a transformar
- Un parámetro `k` que indica el número de vecinos a contemplar si se precisa realizar interpolación. Este parámetro tendrá un valor por defecto de 5.

Cuando se invoca al método `transform` sobre un dato concreto pueden ocurrir dos cosas:

- Que se aplique sobre un dato que sea idéntico a los proporcionados durante el entrenamiento.
- Que se aplique sobre un dato distinto a los vistos durante el entrenamiento.

Si se aplica sobre un dato idéntico a alguno de los proporcionados durante el entrenamiento deberá devolver el *embedding* que tenga calculado.

Si se aplica sobre un dato nuevo, el sistema calculará el *embedding* proporcionado por el *autoencoder* y luego comparará el resultado con los *embeddings* de los patrones de *train* que tendrá almacenados. En dicha comparación, utilizando la distancia euclídea, buscará los `k` más cercanos. Finalmente, devolverá como resultado un promedio de dichos patrones utilizando como peso la distancia a esos `k` patrones más cercanos.

4 La clase Autoencoder

Como se ha descrito previamente, para que la clase `MixedManifoldDetector` pueda realizar su trabajo es necesario pasarle en el constructor un objeto de la clase `Autoencoder`.

La clase `Autoencoder` encapsula en su interior un objeto de la clase `torch.nn.Module`. Este objeto lo usa internamente para implementar el comportamiento del *autoencoder.*

La clase `Autoencoder` debe proporcionar al menos los siguientes métodos para que `MixedManifoldDetector` pueda usarla:

- `__init__(self, epochs : int, error_threshold : float, batch_size : int)`
- `fit(self, data : numpy.ndarray)`
- `transform(self, data : numpy.ndarray)`

4.1 El constructor

El constructor de `Autoencoder` recibirá al menos los siguientes parámetros:

- `epochs` – Un entero con el número de épocas de entrenamiento. Por defecto valdrá 100.
- `error_threshold` – Un flotante con el umbral de error que detiene el entrenamiento aunque no se haya alcanzado el número de épocas prefijado. Por defecto valdrá cero.
- `batch_size` – Un entero con el tamaño del batch que se usará durante el entrenamiento del *autoencoder*.

Opcionalmente, se pueden añadir parámetros relativos a la función de pérdida, el método de optimización, la estructura de la red o los métodos de regularización, aunque nada de esto es obligatorio.

4.2 El método fit

Este método recibirá una matriz de `numpy` con los patrones de entrenamiento. Cada fila corresponderá a un patrón y cada columna a una característica discriminante.

El método `fit` realiza el entrenamiento de la clase.

Internamente, el método `ft` configura la función de pérdida y el método de optimización y ejecuta el bucle de entrenamiento.

4.3 El método transform

Este método recibirá una matriz de `numpy` con patrones. Cada fila corresponderá a un patrón y cada columna a una característica discriminante.

`transform` solo se puede invocar después de haber invocado con éxito al método `fit`. Devuelve los *embeddings* correspondientes a los patrones proporcionados.

5 Experimentos

En los experimentos se desea comprobar el funcionamiento de la clase `MixedAutoencoder` utilizando diferentes combinaciones de *autoencoders* y métodos clásicos de *manifold learning* sobre diferentes bases de datos. Para ello se pide:

- Crear una clase `LinearAutoencoder` que cumpla la interfaz `Autoencoder` y que implemente internamente un modelo con 3 capas lineales en el *encoder*, una capa de 32 neuronas para el *embedding* y 3 capas lineales en el *decoder*.

- Crear una clase `LinearSparseAutoencoder` que cumpla la interfaz `Autoencoder` y que implemente internamente un modelo con regularización *Sparse* y con 3 capas lineales en el *encoder*, una capa de 32 neuronas para el *embedding* y 3 capas lineales en el *decoder*.

- Crear una clase `DenoisingSparseAutoencoder` que cumpla la interfaz `Autoencoder` y que implemente internamente un modelo con regularizaciones *Denoising* y *Sparse* y con 3 capas lineales en el *encoder*, una capa de 32 neuronas para el *embedding* y 3 capas lineales en el *decoder*.

- Probar diferentes configuraciones sobre las bases de datos MNIST y FashionMNIST. Opcionalmente, para obtener la máxima nota, se deberá probar sobre otras bases de datos elegidas por el estudiante (como Cifar10 o Glass Identificaction). Las pruebas incluirán al menos todas las combinaciones de los 3 *autoencoders* definidos y de los algoritmos LLE y TSNE. Asimismo, también incluirá comparativa con LLE y TSNE en solitario (sin autoencoder previo). Las pruebas se ejecutarán sobre datos de *train* y de test.

En el documento PDF que se entregue se espera que contenga:

- Un diagrama de clases del sistema desarrollado, incluyendo todas las clases, métodos y propiedades.

- Las representaciones en 2D, mínimamente comentadas, de todos los experimentos: algoritmos, parámetros...

- Una tabla que resuma los resultados de todas las pruebas realizadas (por ejemplo usando la métrica de *Trustworthiness,* el tiempo empleado en `fit`, o el tiempo usado en `transform`...).