Práctica 2. Convocatoria C4.

Tareas

A continuación se listan las tareas que debes completar en esta práctica. En el enunciado de la convocatoria C2 cuentas con explicaciones más extensas y algunos ejemplos de código, pero ten en cuenta que la numeración de las tareas ha cambiado y hay alguna nueva. Asegúrate de que sigues el guión de tareas de *este* documento. En caso de duda, escribe a tu profesor/a de prácticas. Tus explicaciones en la documentación de cada tarea deben demostrar entendimiento de la cuestión. Cada tarea supone un punto sobre diez, y su documentación supone el 60% de la nota.

1A: Implementa en Python la clase AdaboostBinario y DecisionStump, utilizando Numpy para acelerar los cálculos sobre vectores y matrices. Recuerda que el método DecisionStump.predict() debe devolver +1 o -1, mientras que el método AdaboostBinario.predict() tiene que devolver la suma de los resultados de todos los DecisionStump.predict() multiplicados por sus alfas. Cuando ese valor es mayor que cero, la observación se clasifica como perteneciente a la clase, y cuando no se clasifica como no perteneciente a la clase.

Implementa también una función, que se invoca desde el “main”, que entrene un clasificador AdaboostBinario para cada uno de las diez clases de MNIST y muestre los resultados de cada uno, incluyendo su matriz de confusión.

Recuerda: Para cargar los conjuntos de entrenamiento y de test debes utilizar el siguiente código:

```
(X_train, Y_train), (X_test, Y_test) = keras.datasets.mnist.load_data()
```

Recuerda: Cuando entrenes un AdaboostBinario para una clase (número) concreta, deberás pasarle un conjunto de entrenamiento que tenga la misma cantidad de observaciones de esa clase que de todas las demás (si el número es el 7, deberá haber la misma cantidad de sietes que del resto de número), y la distribución de observaciones de las otras clases debe ser uniforme (es decir, debe tener la misma cantidad de ceros que de unos que de doses, etc.).

1B: Experimenta con los parámetros T y A del método AdaboostBinario.predict, y utiliza la librería matplotlib para mostrar gráficas con la tasa de acierto total y el tiempo de ejecución que obtienes, primero cuando mantienes fijo T y varías A, y segundo cuando mantienes fijo A y varías T. En tercer lugar, explora diferentes combinaciones de T y A para conseguir los mejores resultados en términos de tasa de acierto con la restricción de que en cada combinación que pruebes T*A no debe superar 3600.

Implementa una función, invocada desde el “main”, que ejecute tu experimentación y muestre por pantalla las gráficas con los resultados.

Recuerda: Dado que el entrenamiento de Adaboost no es determinista, diferentes entrenamientos con los mismos datos pueden dar diferentes resultados. Para reducir desviaciones estadísticas debidas a ese indeterminismo, deberás ejecutar el entrenamiento del clasificador varias veces y promediar para obtener cada punto (cada combinación de T y A concreta) de una curva de las gráficas. Cinco ejecuciones por cada punto suele ser suficiente.

Finalmente, recomienda una combinación que maximice la tasa de acierto sin incurrir en sobreentrenamiento ni aplicar más tiempo de ejecución del necesario. Puedes superar T*A=3600 si lo crees conveniente.

1C: Implementa la clase AdaboostMulticlase, cuya función fit cree y entrene un AdaboostBinario para cada clase del conjunto de entrada, y cuya función predict devuelva la clase cuyo clasificador binario haya devuelto un valor más alto.

Implementa una función, que se invoque desde el main, que realice un entrenamiento de tu clasificador multiclase y muestre los resultados, incluyendo la matriz de confusión.

Implementa otra función, que se invoque desde el main, que realice una experimentación similar a la de la tarea 1B, de nuevo utilizando gráficas de matplotlib.

Básate en esos resultados para hacer una recomendación final sobre los valores de T y A, con el objetivo de conseguir la mejor tasa de acierto posible en el conjunto de test para tu clasificador multiclase sin que requiera un tiempo de ejecución excesivo (eso lo valorarás tú en función de tu hardware, pero deberás justificar la decisión en la documentación: recuerda que en aplicaciones reales suele ser más importante conseguir un buen clasificador que ahorrar en el tiempo de entrenamiento).

Recuerda: El conjunto de test con el que medirás la tasa de acierto final de cada clasificador multiclase siempre ha de ser el que devuelve keras.datasets.mnist.load_data(). En algunos casos deberás o podrás modificar el conjunto de entrenamiento (por ejemplo para separarlos en entrenamiento y validación) pero el conjunto de test siempre ha de ser el mismo para todas las técnicas.

1D: Implementa una forma de reducir el número de características (píxeles) que probar en los clasificadores débiles, bien porque reduzcas la dimensionalidad de las imágenes de entrada o bien porque preselecciones los índices de los píxeles que mejor permitan separar clases entre sí. Documenta si este cambio logra mejorar los resultados de tu clasificador multiclase (1C).

Implementa una función invocada desde el main que muestre los resultados de entrenar esta versión de tu Adaboost, para poder comparar con la versión original.

Implementa otra función, que se invoque desde el main, que realice una experimentación similar a la de la tarea 1B, de nuevo utilizando gráficas de matplotlib, pero en este caso jugando con los nuevos parámetros que hayas añadido a en esta versión.

1E: Implementa una versión de tu AdaboostBinario.fit que pare automáticamente el entrenamiento cuando detecte sobreentrenamiento. El conjunto de X_train/Y_train deberás dividirlo en dos, uno de entrenamiento con la mayor parte de las observaciones (imágenes) y otro de validación. El conjunto de validación lo utilizarás para detectar el sobreentrenamiento: cuando al añadir más clasificadores débiles su tasa de acierto comienza a descender.

Implementa una función invocada desde el main que muestre los resultados de entrenar esta versión de tu Adaboost, para poder comparar con la versión original.

Implementa otra función, que se invoque desde el main, que realice una experimentación similar a la de la tarea 1B, de nuevo utilizando gráficas de matplotlib, pero en este caso jugando con diferentes porcentajes en la división entre entrenamiento y validación, con el objetivo de maximizar la tasa de acierto final con el conjunto de test.

Finalmente, comenta las ventajas de este cambio y haz una recomendación de uso del nuevo parámetro.

2A: Implementa una función, que se invoque desde el main, que entrene un clasificador AdaboostClassifier de la librería sklearn utilizando como clasificador débil la clase DecisionTreeClassifier con profundidad 0 (es decir, como vuestro DecisionStump).

Documenta los parámetros más relevantes de ambas clases, centrándote en el caso de DecisionTreeClassifier en los parámetros que son relevantes cuando la profundidad es 0, e identifica qué cambios hay que hacer en esos parámetros para conseguir que se comporte de la misma manera que vuestro DecisionStump: es decir, debe generar en total T clasificadores débiles y probar como mucho A píxeles cada vez que se genera un clasificador débil.

Implementa una función invocada desde el main que muestre los resultados experimentar con esos parámetros buscando la mejor tasa de acierto posible. Si lo crees necesario, puedes crear más de una función, con diferentes configuraciones de parámetros o estrategias de variación de los mismos.

Finalmente, haz una recomendación de valores que permitan la mejor tasa de acierto posible, sin superar el tiempo que requiere tu AdaboostMulticlase original con la configuración que recomiendas al final de la tarea 1B.

2B: Repite la tarea anterior utilizando ahora como clasificador débil árboles de decisión de profundidad mayor que 0.

De nuevo, explica los parámetros de DecisionTreeClassifier, centrándote ahora en los que solo tienen un efecto cuando la profundidad del árbol es mayor que cero.

Experimenta con estos parámetros para mejorar en lo posible la tasa de acierto. Implementa una función invocada desde el main que muestre los resultados de esa experimentación. Si lo crees necesario, puedes crear más de una función, con diferentes configuraciones de parámetros o estrategias de variación de los mismos.

Haz una recomendación final de valores que permitan alcanzar la mejor tasa de acierto posible, sin superar el tiempo que requiere tu AdaboostMulticlase con la configuración que recomiendas en la tarea 1B.

2C: Repite la misma tarea anterior, ahora utilizando la librería Keras para implementar un Multi-Layer Perceptron.

Explica qué hacen, para qué sirven, qué influencia tienen en los resultados, los parámetros más relevantes: número de capas, neuronas por capa, organización de las capas, batch_size, learning_rate, algoritmo de optimización y función de activación.

Experimenta con esos parámetros y trata de mejorar la tasa de acierto que han obtenido las técnicas de las tareas anteriores con el conjunto MNIST. Puedes crear más de una función en el main,

Implementa una función invocada desde el main que muestre los resultados de la experimentación. Si lo crees necesario, puedes crear más de una función, con diferentes configuraciones de parámetros o estrategias de variación de los mismos.

Haz una recomendación final de configuración de tu MLP para resolver MNIST, sin superar el tiempo de ejecución de las técnicas anteriores.

2D: Una vez más, repite la tarea anterior utilizando ahora redes convolucionales (CNN).

Explica el fundamento de este tipo de redes y sus parámetros particulares, y experimenta con ellos para tratar de mejorar los resultados anteriores sin superar su tiempo de ejecución

Implementa una función invocada desde el main que muestre los resultados de esa experimentación. Si lo crees necesario, puedes crear más de una función, con diferentes configuraciones de parámetros o estrategias de variación de los mismos.

Finalmente, haz una recomendación de CNN que obtenga la mejor tasa de acierto posible sin emplear más tiempo que las técnicas anteriores.

2E: Compara los resultados de todos los clasificadores multiclase que has implementado en esta práctica (1C, 1D, 1E, 2A, 2B, 2C, 2D). Si en alguno de esos apartado has desarrollado más de una versión, utiliza la que consideres más efectiva.

Utiliza tablas y gráficos para comparar la eficiencia y efectividad de cada técnica e ilustrar tus conclusiones y tus recomendaciones finales.

Bibliografía

La documentación ha de incluir un apartado de bibliografía donde se referencie cada fuente utilizada: cada libro, página web o herramienta que se haya consultado. En el caso de consultas a chatbots como ChatGPT, se deben incluir los prompts o citar en el texto o código las partes que se han contestado apoyándose en esas herramientas. En el caso de webs de documentación, se deben citar las páginas concretas que se han consultado. En el caso de videotutoriales, se debe incluir un resumen de lo que se ha extraído de esos contenidos. La ausencia de bibliografía penalizará hasta 1 punto. El uso de código o texto copiado entre estudiantes o de fuentes sin referenciar se considerará plagio y significará el suspenso de la práctica.

Fecha y formato de entrega

La fecha límite de entrega de la práctica es el **1 de julio de 2024 a las 23:55h**. La entrega se realizará a través de Moodle.

La entrega debe consistir en **un archivo comprimido zip con un fichero de código (.py) y un fichero en formato PDF con la documentación**. Si has hecho la práctica en Google Colab, adjunta el fichero .ipynb y también un fichero .py donde copies todo el código que ejecutas en tu archivo de Colab.

AVISO IMPORTANTE

No cumplir cualquiera de las normas de entrega descritas en este documento puede suponer un suspenso de la práctica.

Las prácticas son individuales. No se pueden hacer en pareja o grupos.

Cualquier código copiado supondrá un suspenso de la práctica para todos los estudiantes implicados y **se informará a la dirección de la EPS para la toma de medidas oportunas** (Reglamento para la Evaluación de Aprendizajes de la Universidad de Alicante, BOUA 9/12/2015, y documento de Actuación ante copia en pruebas de evaluación de la EPS).