Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Profesor: Dr.-Ing. Pablo Alvarado Moya
Aprendizaje Automático
I Semestre 2024

Entrega: **23 de abril, 2024**

Versión: 27 de marzo de 2024

Proyecto 1: Redes neuronales artificiales y SVM

1. Introducción

En este proyecto utilizaremos bibliotecas estándar en el aprendizaje automático, en una aplicación de reconocimiento de dígitos escritos a mano. En el proyecto no solo buscaremos la forma de entrenar los modelos, sino utilizar métodos estándar en la actualidad, para optimizar los hiperparámetros, para evaluar el desempeño de los modelos, para preparar los datos, etc.

Las bibliotecas seleccionadas para este proyecto son Scikit-Learn y PyTorch usando el lenguaje de programación Python.

Los datos a utilizar corresponden al conjunto MNIST, que contiene alrededor de 70 000 imágenes de 28×28 píxeles con variantes de los dígitos del 0 al 9 escritos a mano (ver figura 1).

Figura 1: Ejemplos de los dígitos en el conjunto de datos MNIST.

La invitación al repositorio del GitHub Classroom con el código base para el proyecto es esta. Recuerde hacer `commit` regularmente para que quede registrado el avance y el responsable de cada cambio en el proyecto.

1.1. Objetivos

Los objetivos del proyecto son:

1. Utilizar redes neuronales y máquinas de soporte vectorial para clasificar dígitos escritos a mano.
2. Aplicar técnicas del aumento de datos para robustecer los procesos de entrenamiento y la capacidad de generalización de los modelos.
3. Realizar el proceso de selección de modelos, para optimizar métricas de clasificación.

4. Utilizar herramientas de protocolo de los procesos de entrenamiento.

5. Aplicar los clasificadores desarrollados en una aplicación demostrativa "en vivo".

2. Gestión del proyecto

Debe utilizar la pizarra de tareas en el proyecto de GitHub (*board*), de modo que se pueda tener siempre bajo control qué persona está trabajando en qué tarea. Puesto que cada tarea (*issue*) tiene asignado un número, utilice ese mismo número en los mensajes de los *commit* para que sea rastreable qué *commit* corresponde a qué tareas en la pizarra.

Estas son buenas prácticas que son comunes hoy en la industria, y aunque son sencillas, requieren cierta práctica para ser incorporadas en la rutina de trabajo. Utilice para esto las columnas "Por hacer", "En progreso", "En revisión" y "Listo". En "Por hacer" se ponen tareas que hay que realizar, pero no se sabe quién o cuándo se van a empezar y se ponen ahí para mantener esas tareas en consideración. En "En progreso" se ponen las tareas que alguien está realizando. Para ello, hay que asignar la tarea a algún miembro (o miembros) del equipo, quien se encarga de realizarla. Una vez lista, se pone en "En revisión". Algún otro miembro del equipo debe revisar el trabajo de la persona (esto se denomina usualmente "*code review*"). Dependiendo de lo que opine el revisor, la tarea se devuelve a "En progreso" para corregir problemas en el código, o a "Listo" en donde terminan las tareas.

Es costumbre para cada una de estas tareas crear una rama de Git, que se mezcla con la rama de desarrollo una vez que llega a "Listo". El profesor durante las clases hará pequeñas sesiones de "*sprint*", en donde revisará el progreso de las tareas semanalmente.

3. Procedimiento

3.1. Instalación

Para este proyecto requiere Python. Es usual, debido a la alta complejidad en el árbol de dependencias de las bibliotecas utilizadas, aislar cada aplicación en entornos, de modo que se puedan instalar las versiones de bibliotecas requeridas en cada caso concreto, si afectar a otras aplicaciones. Por ello, en este proyecto utilizaremos estos entornos.

1. Instalación de Mamba

   El sistema más difundido para el manejo de entornos con control de dependencias es, quizá, anaconda, pero, sin embargo, y debido a su gran popularidad, es un sistema que ha crecido mucho, y en ocaciones se torna demasiado lento. Este sistema se utiliza, por ejemplo, en Google Colab.

   Para la instalación local se recomienda utilizar alternativas más rápidas, como Mamba, que es compatible con conda (el programa utilizado en anaconda para el manejo de paquetes de Python). En particular se recomienda utilizar micromamba, por su facilidad de instalación y su alta velocidad en la resolución de dependencias.

En GNU/Linux, macOS o Git Bash en Windows, se instala con:

```
"${SHELL}" <(curl -L micro.mamba.pm/install.sh)
```

Podemos crear un entorno llamado, por ejemplo, `p1` (por *proyecto 1*), y activarlo, con

```
micromamba create -n p1 python=3.11
micromamba activate p1
```

Para instalar las bibliotecas PyTorch, matplotlib, numpy, etc., si usted tiene acceso a una tarjeta NVidia y tiene CUDA instalado, puede usar las indicaciones en el sitio de PyTorch, donde selecciona lo especificado para `conda`, pero luego lo reemplaza en la línea de comandos por el programa `micromamba`:

```
micromamba install pytorch torchvision torchaudio \
                   pytorch-cuda=11.8 matplotlib numpy \
                   scikit-learn jupyter pillow \
                   -c pytorch -c nvidia
```

Obviamente, usted debe reemplazar la configuración apropiada para su hardware.

Esto usa alrededor de 4 GB de espacio de disco duro, y por tanto tardará descargando los paquetes.

2. Para delegar el entrenamiento y poder computacional a servidores potentes, podemos experimentar con Google Colab, para lo que requerirá una cuenta de google.

   Cada vez que ejecute código allí, necesita re-instalar los paquetes ya sea con `conda` o con `pip`. Es posible utilizar micromamba allí también, pero no es tan simple como hacerlo localmente.

   Para ello, recuerde que si usted precede en Jupyter Notebook a líneas con el signo de admiración '`!`', entonces esa línea será ejecutada en el shell que está ejecutando al jupyter como tal.

3. Además, requerirá hacer una cuenta en Weights & Biases, que es una plataforma comercial para llevar el control de experimentos de entrenamiento, modelos, optimización de hiperparámetros y mucho más. Por suerte, sigue siento aún gratis para estudiantes y académicos.

3.2. Entrenamiento y predicción

En este proyecto usted necesita crear programas para tres etapas distintas: el entrenamiento de los modelos, la optimización de hiperparámetros (o selección de modelos), y la predicción y prueba de los modelos. Usaremos tanto redes neuronales artificiales, usando PyTorch, como máquinas de soporte vectorial, usando Scikit-Learn.

El proyecto tiene una estructura de código más libre que las tareas. Sin embargo, para los clasificadores se brinda una *interfaz* que deben seguir los clasificadores para que puedan

usar el predictor en-vivo implementado en `digitpainter.py` y usado desde `predict.py`. En principio, no es necesario modificar a `digitpainter.py`, y símplemente deberá agregar sus clasificadores en `predict.py`.

Como ejemplo de implementación de un clasificador, el código base tiene listo al softmax, definido en `softmax.py`, que se entrena con `train_softmax.py`, y se evalúa con `test_softmax.py`. Este código no está completo, y debe interpretarse como un ejemplo muy primitivo, que ilustra cómo usar la interfaz mencionada.

Note que el código base entregado está escrito para Python directamente, y no como Python interactivo (Jupyter), porque así se tienen mejores opciones para depuración, y es más fácil llevar un control de versiones con Git "limpio".

Si usted lo prefiere, puede usar cuadernos de Jupyter Notebook o Jupyer Lab, pero debe tener cuidado de no generar conflictos innecesarios con Git, si varias personas suben el mismo cuaderno con los resultados de celdas distintos para cada persona.

1. Estudie el código base entregado para el clasificador softmax, y revise con cuidado cómo se cargan los datos y la estrategia utilizada allí para entrenar y probar el modelo. Esa no es bajo ninguna circunstancia la mejor manera de hacer esta tarea, así que puede utilizar cualquier alternativa que encuentre y considere más apropiada, para los siguientes puntos.

   Compare la forma de hacer este clasificador con respecto a lo implementado en la tarea 4.

2. Usted deberá implementar dos modelos de clasificación supervisada: máquinas de soporte vectorial (SVM) usando Scikit-Learn, y redes neuronales artificiales (ANN) usando PyTorch. Ya el clasificador softmax entregado como ejemplo utiliza ambas bibliotecas.

   Busque en Internet ejemplos sencillos para cada tipo de clasificador (SVM, ANN) entrenándose con MNIST, u otros conjuntos de datos que le permitan a usted aplicar los ejemplos a MNIST.

   Estudie bien los códigos encontrados y comprenda cómo funcionan los procesos de entrenamiento y predicción. Es necesario comprender cómo usa PyTorch los grafos computacionales. A diferencia de GNU/Octave en las tareas 3 y 4, que utiliza un concepto de cálculo de gradientes hacia adelante del paquete correspondiente allí, PyTorch utiliza el concepto explicado en la clase teórica, en donde una fase hacia adelante (`forward`) se usa para calcular el valor de una función, y luego es necesario hacer un proceso hacia atrás para el cálculo del gradiente.

   En PyTorch debe tener cuidado si desean utilizar las aceleraciones utilizando GPU, puesto que los tensores utilizados deben transferirise al hardware deseado. En el entrenamiento y predicción del softmax se ejemplifica cómo hacer esto. Además, debe tenerse cuidado cuándo re-inicializar el cálculo del gradiente, y cuando debe desactivarse por completo. ¡Esto último es fuente muy común de errores en PyTorch! ¡Tenga mucho cuidado con eso!

3. Adapte los códigos encontrados para la red neuronal y para SVM, de modo que sigan la interfaz especificada a través de `classifier.py`. Se espera entonces que cree los archivos `ann.py` y `svm.py` con los modelos de red neuronal artificial y máquina de soporte vectorial, respectivamente.

   Puesto que más adelante usted deberá evaluar distintas configuraciones de los modelos, deberá encontrar alguna forma de especificar características particulares de los mismos, como el tamaño de las capas, las capas de activación, los algoritmos de optimización usados, las funciones de pérdida, etc.

   Aquí el uso de diccionarios y la sintaxis con doble asterisco '`**`' para pasar el contenido del diccionario como argumentos a una función, puede serle de mucha utilidad

4. Cree entonces los archivos `train_ann.py` y `train_svm.py` para entrenar la red neuronal y la máquina de soporte vectorial. Dicho código debe guardar los modelos entrenados en un archivo, utilizando el método `save` definido por la interfaz.

5. Para el caso de la red neuronal, investigue sobre el aumento de datos (ver por ejemplo esto) de modo que usted pueda modificar levemente, con traslaciones y escalamientos aleatorios, a los datos originales. Esto pretende mejorar la capacidad de generalización del modelo a datos reales en el siguiente punto, pues el conjunto de datos en general ajusta los dígitos a posiciones y escalas relativamente rígidas, y cuando un usuario real dibuja un dígito, usualmente no seguirá esos estándares.

   En su código usted deberá poder activar y desactivar fácilmente el uso del aumento de datos, para poder evaluarlo posteriormente. ¡No exagere con el aumento de datos! Debe producir modificaciones que sean realistas con lo que una persona puede ingresar en el programa de dibujo.

6. Escriba los archivos `test_ann.py` y `test_svm.py` que evalúen con las métricas básicas de precisión y exhaustividad a los modelos y que presenten las matrices de confusión. Observe que estos archivos deben usar el método `load` definido por la interfaz para cargar el archivo almacenado en los archivos de entrenamiento.

7. Agregue sus clasificadores en `predict.py`. Si usted siguió de forma consistente el diseño de la interfaz, esto debería consistir en agregar una línea adicional para cada clasificador. Este predictor va a utilizar el método `predict` definido por la interfaz para encontrar el modelo.

Los resultados por usted obtenidos hasta aquí consistirán en su *línea base* que deberá ser documentada en sus resultados: la configuración concreta de los clasificadores usados, así como los valores de precisión, exhaustividad y pérdida alcanzados. Los siguiente pasos buscarán mejorar estos resultados.

3.3. Selección de modelos

En los procesos de diseño de modelos para el reconocimiento de patrones, la *selección de modelos* se encarga de seleccionar los hiperparámetros que permiten llegar a un buen

modelo. Hay dos tipos de hiperparámetros: los que configuran al modelo mismo, y los que configuran al proceso de entrenamiento. Ambos son relevantes para el resultado final.

Estos procesos de selección de modelos requieren mucho tiempo porque involucran entrenamientos con cada configuración evaluada, y requiere de un buen protocolo experimental para llegar a las configuraciones deseadas.

Hay mecanismos sistemáticos de evaluación, como la búsqueda de rejilla (*grid search*) en la que se barren parámetros en rangos de valores, que sin embargo toman mucho tiempo de ejecución, y se desperdicia tiempo y energía evaluando configuraciones que no tienen mucho sentido. Por otro lado, existen otros mecanismos más modernos, basados en marcos probabilísticos robustos, que buscan minimizar el número de entrenamientos para encontrar buenas configuraciones. Una buena introducción al tema la encuentra aquí.

1. Copie sus archivos de entrenamiento, de modo que en una copia usted agregue lo necesario para utilizar Weights & Biases, y protocolar cada experimento que usted realice, de modo que posteriormente pueda comparar todos los experimentos que quiera.

   Para el caso de la red neuronal, es importante que protocole la evolución de la pérdida en función de las iteraciones o épocas, tanto con los datos de entrenamiento como con los de prueba, de modo que pueda luego realizar los diagnósiticos básicos de sobreajuste, sesgo del modelo, etc.

2. Modifique sus nuevos archivos para buscar los hiperparámetros que usted considere convenientes de optimizar, usando el mecanismo de barrido bayesiano mencionado anteriormente. Usted debe encontrar la manera de almacenar las configuraciones utilizadas con los valores de desempeño alcanzados, de modo que en puntos posteriores usted pueda realizar más análisis del desempeño de esos clasificadores.

   Para la red neuronal, usted deberá evaluar varias configuraciones, variando el número, tipo y tamaño de las capas de combinación (convolucionales o totalmente conectadas), capas de regularización (dropout), capas de activación (ReLU, PreLU, etc.), algoritmos de optimización (Adam, Nadam, SGD, etc.). Un buen diseño del código le permitirá hacer estas evaluaciones de forma bastante límpia, así que debe investigar cómo hacer esto bien.

3. Con el SVM deberá evaluar el tipo de kernel utilizado, y los valores del parámetro C y eventualmente algún parámetro propio del kernel utilizado.

4. Un problema que tienen los mecanismos anteriores es que evalúan una única métrica, como la pérdida con el valor F_1, y eso en ocasiones conduce a resultados erróneos o sesgados. Entonces, para las configuraciones almacendas en los puntos anteriores, usted deberá realizar más análisis de su desempeño.

   Para esto necesitamos explicar primero el concepto de optimalidad Pareto. Este concepto se aplica cuando se tienen múltiples objetivos a optimizar simultáneamente. Para comprenderlo es necesario primero aclarar el término de dominancia Pareto.

Sea $\underline{\mathbf{x}}$ un vector de medidas de aptitud. Por ejemplo, $\underline{\mathbf{x}} = [x_1, x_2]^T$ con x_1 la precisión y x_2 la exhaustividad, tal que mientras más alto el valor de estas aptitudes, mejor se considera su desempeño. Se dice que el vector $\underline{\mathbf{x}}$ domina al vector $\underline{\mathbf{y}}$ si en todas las aptitudes se cumple $x_i \geq y_i$ y en al menos una de ellas $x_i > y_i$, aunque con frecuencia se aproxima símplemente con $x_i > y_i$ para todas las aptitudes.

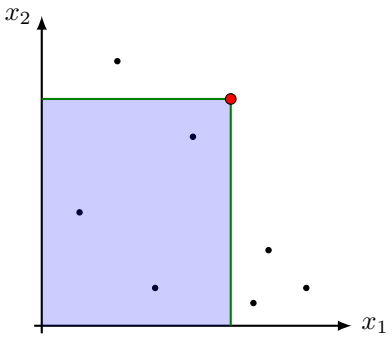

Figura 2: Rectángulo sombreado denota el área dominada por el punto rojo. Hay cuatro puntos no dominados y tres puntos dominados.

Gráficamente, se denota al rectángulo (o hiperprisma rectangular en más dimensiones) entre el origen y $\underline{\mathbf{x}}$, como el área (o volumen) dominada por $\underline{\mathbf{x}}$, como lo ilustra la figura 2.

El frente de Pareto de un conjunto de vectores $\underline{\mathbf{x}}$ lo conforman todos aquellos vectores que no están dominados por ningún otro vector del conjunto.

Por ejemplo, en la figura 3 se tienen ocho puntos. Cada punto tiene marcada el área que domina. Solo los cuatro puntos marcados con rojo no están dominados y ellos conforman entonces al *frente de Pareto* de este conjunto de ocho puntos. Cualquier punto en el frente es óptimo en el sentido de que no existe ningún otro punto que sea mejor en todas las aptitudes. Todos los puntos debajo del frente de Pareto no son óptimos porque están dominados por al menos uno de los puntos en el frente.

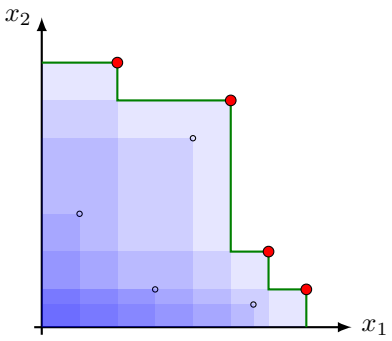

Figura 3: Frente de Pareto conformado por los puntos rojos. Toda el área dominada está bajo la linea verde. Cualquier punto en el frente es óptimo, en el sentido de que no hay ningún otro punto mejor que todas las aptitudes.

Si usted tiene pares precisión/exhaustividad almacenados como filas de una matriz,

entonces la siguiente función[1] permite encontrar un vector de booleanos, en el que serán `True` solo aquellas entradas correspondientes a puntos no dominados; es decir, dicho vector indicaría cuáles son los puntos correspondientes al frente de Pareto.

```
# For each data point compute if it belongs to the Pareto front
def is_pareto_optimal(fitness):
  """
  Find the pareto-optimal points
  :param fitness: An (n_points, n_fitness) array (each data in a row)
  :return: A (n_points, ) boolean array, indicating whether
           each point is Pareto optimal
  """
  is_efficient = np.ones(fitness.shape[0], dtype = bool)
  for i, c in enumerate(fitness):
    if is_efficient[i]:
      is_efficient[is_efficient] = np.any(fitness[is_efficient]>c,
                                          axis=1)
      is_efficient[i] = True  # And keep self
  return is_efficient
```

Escriba una función entonces que utilice esto para encontrar y mostrar el espacio precisión/exhaustividad para los clasificadores seleccionados, así como aquellos puntos que se encuentran en el frente de Pareto, de forma similar a lo ilustrado en la figura 4.

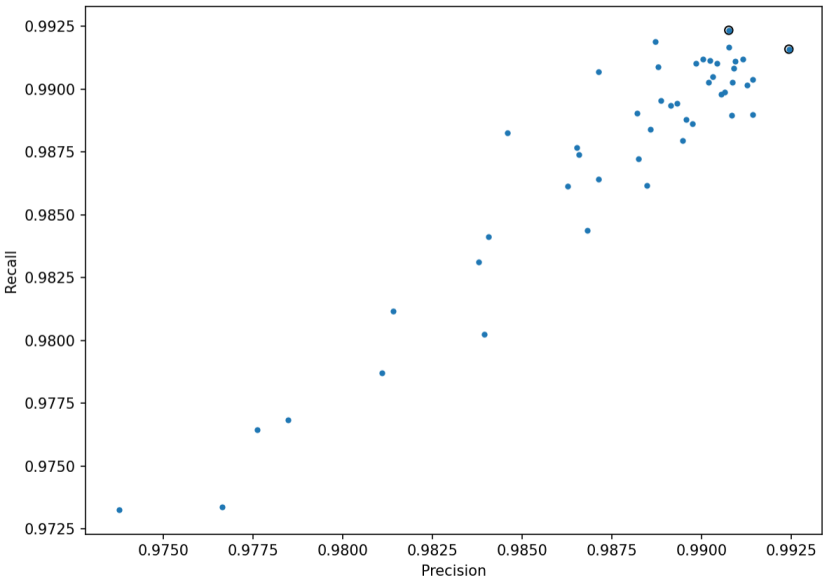

Figura 4: Ejemplo de evaluación de varios hiperparámetros, y aquellos puntos en el frente de Pareto.

5. Para cada clasificador, elija una de las configuraciones en el frente de Pareto y

[1] Tomado originalmente de este sitio.

entrene el modelo con todo el set de entrenamiento. Almacene esos modelos para su uso con el predictor.

6. Para el mejor modelo de cada clasificador presente la matriz de confusión y el reporte de métricas por clase, evaluando con el conjunto de datos de validación.

7. Diseñe algún experimento para evaluar qué efecto tiene el aumento de datos en reconocer dígitos escritos a mano en la aplicación entregada. Para ello deberá captuar algunos datos por su cuenta, modificando levemente la aplicación entregada para ese fin. Por ejemplo, puede hacer un clasificador falso, pero que sigue la interfaz, y cada vez que se llama, guarda una nueva imagen.

   Para esta recolección de datos, será bueno que todos los grupos colaboren con dígitos dibujados por todos los miembros, de modo que creemos un set estándar de prueba.

4. Entregables

1. Archivos de Python.
2. Archivo README con instrucciones de cómo ejecutar su código.
3. El archivo anterior deberá indicar de dónde descargar los mejores modelos entrenados para usar con el predictor.
4. Archivo PDF con las gráficas generadas y el análisis de resultados.
5. Favor subir la última versión del código al tecDigital.
6. El trabajo en Git se tomará en cuenta y la versión en la rama `master` o `main` deberá corresponder con la versión entregada. Marque esa versión con un `tag` adecuado.

5. Notas

- Este proyecto se debe resolver entre tríos o cuartetos, correspondientes a la unión de grupos de trabajo de las tareas.
- Es necesaria la revisión detallada de la teoría del curso para poder resolver y comprender los subproblemas a resolver.
- Los archivos con el mejor modelo pueden ser demasiado grandes para el repositorio Git o el tecDigital, por lo que puede compartirlos con el profesor a través de Google Drive, Dropbox, Mega, OneDrive, o servicios similares.
- Puesto que el entrenamiento tarda bastante y no siempre las versiones que tenga el profesor coinciden con las utilizadas, todos los resultados deben entregarse en el archivo pdf (la curvas y algunas matrices de confusión que considere relevante mostrar, para distintos hiperparámetros). De nuevo, recuerde que en Colab tiene a disposición bastante poder computacional.

Toda tarea debe ser resultado del trabajo intelectual propio de la persona o personas que la entregan. Además de la literatura de referencia, solo puede utilizarse el material expresamente así indicado en la tarea, lo que incluye código brindado por el profesor o indicado en los enunciados a ser utilizado como base de la tarea. Expresamente quedan excluidos como material de referencia los trabajos entregados por otros estudiantes en el mismo semestre o semestres anteriores.

Nótese que esto no elimina la posibilidad de discutir estrategias de solución o ideas entre personas y grupos, lo cual es incluso recomendado, pero la generación concreta de cada solución, derivación o programa debe hacerse para cada entrega de forma independiente.

Para toda referencia de código o bibliografía externa deben respetarse los derechos de autor, indicando expresamente de dónde se tomó código, derivaciones, etc. Obsérvese que el código entregado por el profesor usualmente ya incluye encabezados con la autoría correspondiente. Si un estudiante agrega código a un archivo, debe agregar su nombre a los encabezados si la modificación es de más del 50 % del archivo, o indicar expresamente en el código, con comentarios claros, la autoría de las nuevas líneas de código, pues a la autora o al autor del archivo original no se le debe atribuir código que no es suyo.

Si se detecta código o deducciones teóricas iguales o muy cercanas a trabajos de otros estudiantes del mismo semestre o de semestres anteriores, se aplicará lo establecido por la reglamentación vigente, en particular el Artículo 75 del Reglamento de Régimen de Enseñanza y Aprendizaje.

Modificaciones de comentarios, cadenas alfanuméricas, nombres de variables, orden de estructuras independientes, y otras modificaciones menores de código se siguen considerando como clones de código, y las herramientas automatizadas de detección reportarán la similitud correspondiente.

Los estudiantes que provean a otros estudiantes del mismo o futuros semestres soluciones de sus tareas, también son sujetos a las sanciones especificadas en la reglamentación institucional. Por lo tanto, se advierte no poner a disposición soluciones de las tareas a otros estudiantes.