TAREA 2: Aprendizaje supervisado

Profesor: Fernando Crema Garcia
Fecha: 18 de Febrero 2026

La nota final será basada en la puntuación de clasificación ponderada por un coeficiente **extra** dependiendo de la calidad del reporte asociado.

Clasificación	Extra
C	α

Fecha de entrega: 02/03/2026 23:59 pm Caracas.
Penalizaciones: 2 ptos / día. Máximo 10 ptos.
Nota final:
$$\text{mín}\left\{20\alpha\left(\frac{C}{\text{puntos totales}}\right), 20\right\}$$

I. Introducción

I-A. Objetivos

El objetivo de esta actividad es reforzar las habilidades adquiridas hasta el momento para aprendizaje supervisado específicamente para clasificación, comparando los modelos vistos en clases:

1. Regresión Logística
2. Clasificación con k-vecinos (KNN)
3. Máquinas de soporte vectorial (SVM)
4. Árboles de Decisión

Siempre asumiendo que, para cada caso, aplicamos correctamente los conocimientos de preprocesamiento, selección de modelos y pipeline de proyectos en aprendizaje supervisado. Además, el estudiante dispone de una aplicación web en **Reflex** que prueba los modelos entrenados.

II. Evaluación

II-A. Reproducibilidad

Cada una de las secciones tiene la misma ponderación y al final será la suma de todos los ejercicios que lograron resolver ponderada por un criterio del grupo docente (α).

Para lograr que el grupo docente reproduzca sus resultados, deben asignar al comienzo de su entregable la semilla referente a su **cédula de identidad**

II-B. Modelos entrenados

Sin importar la sección de la tarea que esté realizando, asuma que debe agregar un notebook **por sección** en el cual deben explicar toda la lógica de entrenamiento de los modelos desde preprocesamiento de datos, selección de variables, selección de hiper parámetros y evaluación de modelos. Se deja a juicio del estudiante agregar tanta información sea necesaria para justificar sus hallazgos (gráficos, tabla de resultados, etcétera) asumiendo que el Notebook va a ser corrido en un ambiente independiente [1].

De igual manera, se deben respetar las siguientes restricciones:

1. Un Jupyter Notebook con uso adecuado de los ítems de headings (#) separando cada nivel de acuerdo al tipo de modelo y nombre del mismo. Esta vez, se deja a juicio de ustedes la configuración del mismo.
2. Asuman que el Notebook lo va a leer una persona del equipo de toma de decisiones de una empresa pero que tiene conocimiento previo en Aprendizaje Automático. Sean tan detallados como sea posible justificando cada decisión. **No hay soluciones únicas**.
3. Incentivo la creatividad de ustedes y formará parte de la evaluación de α
4. **No se pueden usar soluciones de terceros de manera directa**. Nunca es un problema revisar soluciones de terceros. Sin embargo, su solución debe ser producto de **su** trabajo. Plagios serán penalizados con nota mínima.
5. Incentivo la colaboración como descrito en el **código de honor** de la materia. Sin embargo, copias serán penalizadas con nota mínima.

II-C. Aplicación de Reflex (Clasificación)

En el caso específico de la parte de clasificación se espera el uso de la aplicación en Reflex siguiendo los pasos del repositorio base proporcionado. La aplicación ya cuenta con la estructura necesaria: páginas, componentes, estados y utilidades. El trabajo del estudiante consiste en entrenar los modelos y conectarlos a la aplicación. Si el estudiante decide extender la aplicación considerando una parte donde podamos ver la comparación entre los distintos modelos será considerado en el α extra de su nota aunque no es obligatorio.

email: fernando.cremagarcia@kuleuven.be

[1] Esto significa que no puede depender de archivos externos, si tienen alguna duda de cómo lograr esto contacte al grupo docente

III. Clasificación

III-A. Objetivo

El objetivo principal de esta parte es utilizar la sección `A jugar` donde tienen un panel como el siguiente:

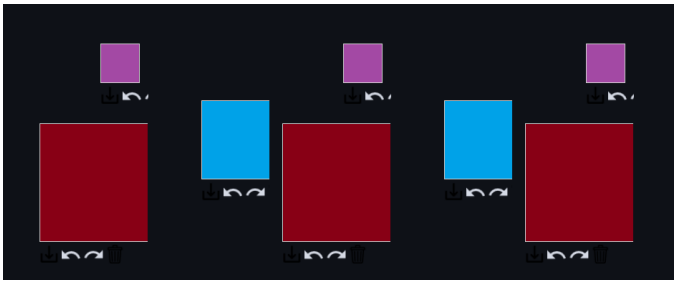

Figura 1. Canvas de imagen donde los cuadrados rojos (base) y magenta (exponentes) reciben números y los cuadrados color azul reciben Operadores. Se deben realizar en total 8 predicciones y luego ejecutar la operación $base_1^{exponente_1}\ operador_1\ base_2^{exponente_2}\ operador_2\ base_3^{exponente_3} =$ resultado

en el cuál pueden ejecutar una operación matemática sencilla y evaluar su resultado.

En el notebook asociado a esta sección, debe comparar exhaustivamente los cuatro modelos vistos en clases hasta el momento: regresión logística, k-vecinos, SVM y árboles de decisión. Dependiendo de sus resultados, escoja y almacene uno de ellos [2] y úselo en la aplicación para ejecutar las operaciones.

La parte crítica de la tarea es la comparación de los modelos por lo que el uso de figuras y tablas es obligatorio [3]. Además, recuerde que existen para cada modelo distintas técnicas de regularización para evadir sobreajuste por lo que **deben ser consideradas en todos los modelos**

El canvas es la sección de nuestra aplicación de Reflex que da uso al modelo escogido en la sección anterior.[4]

Tenemos entonces tres tipos de *input* en nuestro *canvas*:

1. Exponentes: 3 posibles referentes a los cuadrados morados. Deben ser números del 0 al 9.
2. Operadores: 2 posibles referentes a los cuadrados azules. Explicados en la siguiente sección.
3. Números: 3 posibles referentes a los cuadrados rojos. Deben ser números del 0 al 9.

[2]Si así desea puede cargar tantos modelos como quiera en la aplicación siempre que sea natural el proceso de selección dentro de la aplicación. Por comodidad, solo es necesario usar uno.

[3]Es recomendado dividir el trabajo al menos en tres secciones: metodología, resultados y discusión. Tanto para el preprocesamiento y transformación de los datos como para el entrenamiento de los modelos

[4]Si bien incentivamos el desarrollo de una aplicación usable, robusta, eficiente y eficaz recuerden que lo más importante es cubrir los objetivos asociados al aprendizaje automático. Esta sección es para divertirse y usar una pequeña aplicación. La creatividad queda del lado de ustedes.

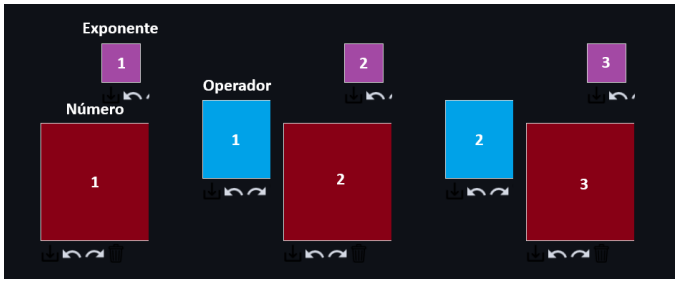

Figura 2. Canvas de imagen

III-B. Modelo de operadores

III-B1. Datos de entrada: El dataset **debe** ser creado por ustedes! La cantidad de imágenes, estilo, resolución y cualquier elemento que consideren relevante debe ser decidido por ustedes: sean creativos. En el `notebook` referente al modelo de operadores, es necesaria una sección que explique toda la toma de decisiones.

III-B2. Clases del modelo: Solo vamos a usar las 4 operaciones fundamentales: suma, resta, multiplicación y división.

En el caso de suma y resta las únicas opciones posibles son: + (ASCII Code 43) y - (ASCII Code 45), respectivamente.

En el caso de multiplicación y división tendremos 2 opciones como sigue:

III-B2a. Multiplicación: Una × (ASCII Code 215) o un asterisco * (ASCII Code 42)

III-B2b. División: Un `slash` / (ASCII Code 47) o el operando convencional ÷ (ASCII Code 247)

III-C. Modelo de clasificación de números

III-C1. Datos de entrada: Existen muchísimas opciones para leer los datos de MNIST. Por ejemplo, el siguiente código ya está disponible en la aplicación del repositorio:

```
from keras.datasets import mnist
from functools import lru_cache

@lru_cache(maxsize=1)
def get_mnist_data():
    return mnist.load_data()
```

Esta sección de código usa el dataset mnist con la interfaz de Keras para facilitar consistencia en todos los proyectos.

En los notebooks del curso hemos usado el paquete torchvision en el caso de usar modelos de redes neuronales.

```
from torchvision.datasets import MNIST

train_dataset = MNIST(
  root='../data',
  train=True,
  transform=torchvision.transforms.ToTensor(),
  download=True
```

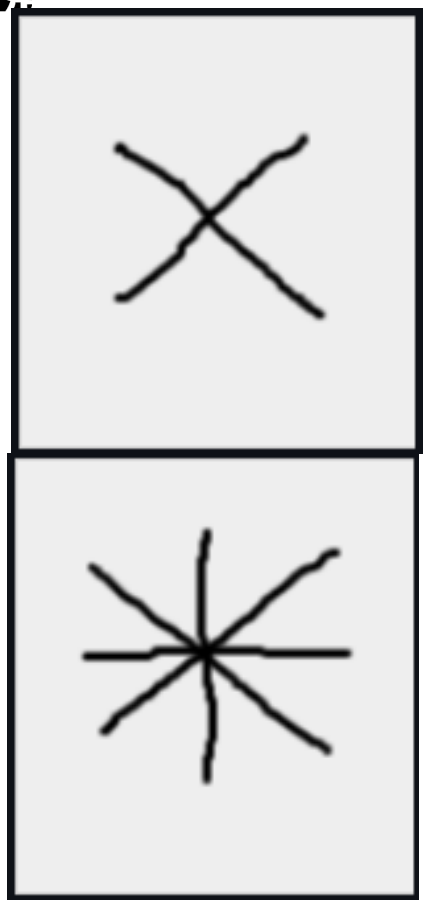

Figura 3. Multiplicación

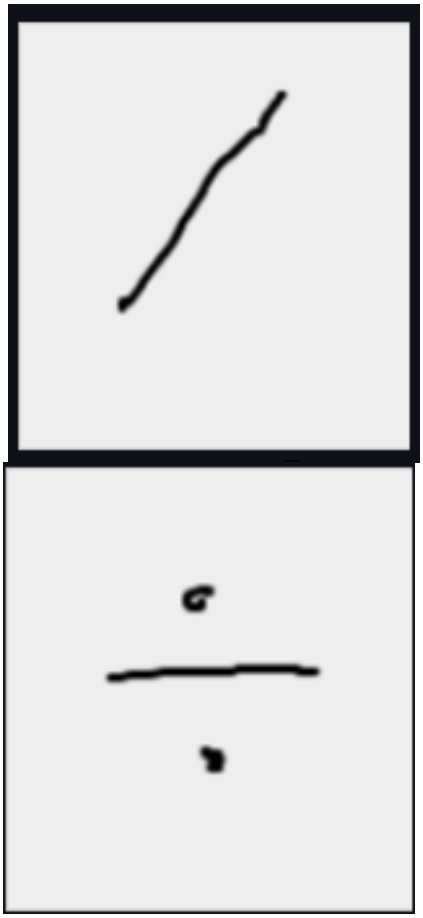

Figura 4. División

```
)
test_dataset = MNIST(
  root='../data',
  train=False,
  transform=torchvision.transforms.ToTensor(),
  download=True
)
```

Existen muchas maneras de usar el dataset y el diseño de su solución depende de ustedes. Sea lo más explícito posible en cada paso y decisión.

III-C2. Sobre la evaluación del pipeline:

1. En la sección `notebooks` deben crear uno o dos archivos `.ipynb` donde expliquen claramente todo el `pipeline` que usaron para crear los modelos, explicar el proceso de preprocesamiento, creación de datos (de ser necesario) y cualquier elemento que consideren relevante.
2. Los modelos asociados a la solución principal deben haber sido vistos en clase.
3. Puede usar modelos **extras** para comparar rendimiento aún cuando no hayan sido vistos en clases.

III-D. Comentarios

III-D1. Sobre el framework de la aplicación: El código base está hecho en Reflex, un *framework* para desarrollar aplicaciones web completas en Python.

La Documentación de Reflex es bastante sencilla de entender y la mayoría de funcionalidades necesarias ya están implementadas en el repositorio base.

III-D2. Cómo ejecutar la aplicación:

III-D2a. Desarrollo local:

1. `pip install -r requirements.txt` instala las dependencias.
2. `reflex init` inicializa el proyecto.
3. `reflex run` ejecuta la aplicación en `http://localhost:3000`.

III-D2b. Docker:

- `docker compose build` crea el contenedor.
- `docker compose up` lo ejecuta en modo desarrollador.
- `docker compose up -d` lo ejecuta en modo `daemon`.

III-D3. Sobre las operaciones:

1. Asumimos que la aplicación siempre será usada por un agente honesto. No se debe validar para datos que no sean los referentes al modelo (aunque es un problema interesante de resolver).

2. Somos consistentes en la entrada de cada `canvas` así como en el orden de las operaciones: de izquierda a derecha y con prioridad de operadores: ^, (*, /), (+, -).

III-D4. Sobre la parte visual: La aplicación ya cuenta con las siguientes páginas:

- **Inicio** (`/`): Página principal con instrucciones.
- **A Jugar** (`/jugar`): Panel con 8 canvas (3 números, 3 exponentes, 2 operadores) para evaluar expresiones.
- **MNIST** (`/canvas-demo`): Explorador del dataset MNIST.
- **Cargar Modelos** (`/cargar-modelos`): Carga de modelos ML (`.joblib`, `.pickle`, `.keras`).

El trabajo del estudiante consiste en entrenar los modelos, exportarlos y cargarlos en la aplicación para habilitar la funcionalidad de `A Jugar`.

III-D5. Sobre la exportación de modelos: Una vez entrenado el modelo en el notebook, se debe guardar para luego cargarlo en la aplicación. Por ejemplo, con `joblib`:

```
import joblib

# modelo = ...  (tu modelo ya entrenado)
joblib.dump(modelo, "mi_modelo_digitos.joblib")
```

Los formatos soportados son:

- **Joblib** (`.joblib`): `joblib.dump(modelo, "archivo.joblib")`
- **Pickle** (`.pkl`, `.pickle`): `joblib.dump(modelo, "archivo.pkl")`
- **Keras** (`.h5`, `.keras`): `modelo.save("archivo.h5")`

Cualquier clasificador de scikit-learn funciona sin modificar el código de la aplicación (KNN, SVC, LinearSVC, RandomForest, etc.).

III-D6. Sobre el preprocesamiento en la app: Es importante que el preprocesamiento aplicado a las imágenes del canvas coincida **exactamente** con lo que se hizo durante el entrenamiento. Para ello, editar el archivo `math_recognizer/utils/image_processing.py` según corresponda.

III-D7. Estructura del proyecto:

```
math_recognizer/        # Paquete principal (Reflex)
  state/                # Clases de estado
  components/           # Componentes reutilizables
  pages/                # Páginas de la aplicación
  utils/                # Utilidades (procesamiento de imagen)
assets/                 # Recursos estáticos (imágenes, JS)
models/                 # Modelos ML (input/output)
notebooks/              # Notebooks de documentación y análisis
```