Universidad de
SanAndrés

I302 - Aprendizaje Automático y Aprendizaje Profundo

2do Semestre 2024

Trabajo Práctico 4

Fecha de entrega: Miércoles 23 de octubre, 23:59 hs.

Formato de entrega: Los archivos desarrollados deben ser entregados en un archivo comprimido .zip a través del Campus Virtual, utilizando el siguiente formato de nombre de archivo: *Apellido_Nombre_TP4.zip*. Se aceptará únicamente 1 archivo por estudiante. En caso de que el nombre del archivo no cumpla con la nomenclatura especificada, el trabajo no será corregido.

Dentro del archivo .zip, debe incluirse un Jupyter Notebook llamado *Entrega_TP4.ipynb* con las respuestas a los problemas y los gráficos resultantes. Puede agregar resultados o análisis adicionales si lo considera necesario. Se recomienda fuertemente no realizar todo el desarrollo dentro del Jupyter Notebook; en su lugar, se sugiere usar archivos .py para desarrollar el código, siguiendo las buenas prácticas de programación y modularización vistas en clase.

Se recomienda seguir la estructura de archivos sugerida al final del trabajo práctico.

Trabajo Práctico 4: Redes Neuronales

El objetivo de este trabajo es implementar y entrenar una red neuronal para estimar el precio de venta de una SUV (Sport Utility Vehicle) de Toyota. El dataset *Precios de SUVs* contiene información sobre los precios en dólares de tres tipos de vehículos SUV de la marca Toyota, publicados en la página de Mercado Libre Argentina, junto con características adicionales relevantes de cada publicación. Previamente, el conjunto de datos fue dividido en subconjuntos de desarrollo (*toyota_dev.csv*) y prueba (*toyota_test.csv*), los cuales deberán ser utilizados para entrenar y evaluar los modelos.

NOTA: Use exclusivamente NumPy para la implementación de funciones y/o clases. No se permite el uso de librerías de Machine Learning como Scikit-Learn, TensorFlow o PyTorch, salvo indicación específica en la consigna.

1. Implementar una red neuronal multicapa densa ("fully connected") con $L \geq 1$ capas ocultas (profundidad de la red) y $M^{(\ell)} \geq 1$ neuronas por capa (ancho de la capa ℓ). Deberán implementar el algoritmo de descenso por gradiente junto con backpropagation para optimizar los pesos de la red, utilizando como función de pérdida la suma de los errores cuadráticos.

   a) Dividir el conjunto de desarrollo en dos subconjuntos: uno para entrenamiento (80 %) y otro para validación (20 %). Entrenar una red neuronal con $L = 3$ capas ocultas y $M^{(1)} = 10$, $M^{(2)} = 8$, $M^{(3)} = 4$ neuronas en cada capa. Usar ReLU como función de activación en las capas ocultas y un learning rate de 1×10^{-8}. Reportar RMSE, MAE y R^2 en validación y graficar las curvas de error de entrenamiento y validación.

2. Para la red neuronal multicapa densa implementada en el inciso *a*), deberá implementar un *Learning Rate Scheduler*, que permita ajustar dinámicamente el learning rate durante el entrenamiento, en lugar de utilizar un valor constante. Explorar las siguientes variantes, comparando el rendimiento de la red para cada una y graficando la evolución del error en función de las épocas:

   a) Linear decay

   b) Power law

   c) Exponential decay

   Sugerencia: Para más detalles sobre Learning Rate Scheduler, ver Capítulo 7.3.2. de Bishop (2023).

3. Para la red neuronal multicapa densa implementada en el inciso *a*), implementar la(s) siguiente(s) técnica(s) para evitar el sobreajuste:

   - *Regularización L2.* Ajustar el hiperparámetro de regularización λ utilizando el conjunto de validación. Mostrar los gráficos de error de entrenamiento y validación para distintos valores de λ.

- OPCIONAL: *Dropout* en las capas ocultas. Ajustar la tasa de dropout p utilizando el conjunto de validación. Deberán mostrarse los gráficos de error de entrenamiento y validación para distintos valores de p.
- OPCIONAL: *Early Stopping* para detener el entrenamiento cuando el error de validación deje de mejorar tras un número predefinido de épocas consecutivas. Ajustar los hiperparámetros utilizando el conjunto de validación. Deberán mostrarse los gráficos de error de entrenamiento y validación para distintos valores de los hiperparámetros.

  Sugerencia: Para más detalles sobre Dropout, ver Capítulo 9.6.1 de Bishop (2023) y 13.5.4 de Murphy (2022). Para Early Stopping, consultar Capítulo 9.3.1 de Bishop (2023) y 4.5.6 de Murphy (2022).

4. Para la red neuronal multicapa densa implementada en el inciso *a*), deberá implementar los siguientes métodos de optimización. Asegúrese de que cada método esté implementado de manera modular para que se pueda seleccionar fácilmente cuál usar en el proceso de entrenamiento:

   - Descenso por gradiente estocástico.
   - Descenso por gradiente estocástico con momentum.
   - Descenso por gradiente con mini-batches.
   - Adam.

   a) Ajustar los hiperparámetros relevantes para cada optimizador y justificar su elección en base al rendimiento obtenido en términos de error de validación y velocidad de convergencia.

   b) Graficar en dos figuras separadas la evolución del error en función de las épocas para cada método de optimización, una con el error de entrenamiento y otra con el error de validación. Asegúrese de incluir leyendas claras y etiquetas en los ejes para facilitar la comparación entre los métodos en ambas figuras.

   Sugerencia: Ver los Capítulos 7.2.3, 7.2.4, 7.3.1 y 7.3.3 de Bishop (2023) para detalles sobre la implementación y ajuste de cada optimizador.

5. Utilizando el conjunto de desarrollo y la metodología de cross-validation con 5 folds, seleccionar los hiperparámetros L y $M^{(\ell)}$ óptimos para la red neuronal. Para ello, puede utilizar la inicialización de parámetros que considere apropiada y el optimizador que usted desee (normalmente, uno querrá usar el optimizador que mejor resultado le haya dado, en cuanto a performance del modelo entrenado y el tiempo de convergencia).

   Sugerencia: Si bien uno puede hacer una búsqueda exhaustiva sobre el espacio de hiperparámetros L y $M^{(\ell)}$, esto puede resultar computacionalmente costoso, debido a que para cada combinación y cada fold, se debe re-entrenar el modelo y evaluarlo.

Piense si existe alguna manera de recorrer el espacio de búsqueda con menor costo computacional. Además, recuerde que dados dos modelos de performance similar, se privilegia elegir el de menor complejidad (menor cantidad de parámetros), porque tenderá a tener menor varianza en el error predictivo sobre datasets "nuevos" (es decir, que no hayan sido usados para el desarrollo del modelo).

6. Evaluar la performance del modelo final implementado en el inciso e) siguiendo los criterios detallados a continuación. Realice un análisis breve de los resultados, explicando cómo varían las métricas entre los diferentes conjuntos de datos.

   a) Reportar los valores de RMSE_{CV}, MAE_{CV} y ${\text{R}^2}_{\text{CV}}$ obtenidos sobre los datos "held out" de los folds utilizados en la validación cruzada.

   b) Evaluar el rendimiento del modelo sobre el conjunto de prueba para obtener una estimación del rendimiento general en datos no vistos. Reportar los valores de $\text{RMSE}_{\text{test}}$, MAE_{test} y ${\text{R}^2}_{\text{test}}$ obtenidos.

   c) Para cada conjunto de datos ("held out" de CV y test), grafique el histograma de los residuos absolutos. Seleccione un número de bins adecuado que capture la tendencia general de los residuos, evitando representaciones demasiado erráticas.

   d) Para cada conjunto de datos ("held out" de CV y test), graficar el diagrama de dispersión de las predicciones $\hat{y}$ frente a los valores verdaderos y.

   e) Comparar la performance de la red neuronal con la de los modelos de regresión lineal, regresión localmente ponderada y regresión no lineal desarrollados en el Trabajo Práctico 2. Indique cuál de los cuatro modelos cree que generalizará mejor y justifique su elección.

7. Implementar el modelo final del inciso e) utilizando PyTorch. Compare el rendimiento de este modelo con la implementación previa en términos de tiempo de entrenamiento y métricas de evaluación.

Estructura Sugerida para la Entrega del Trabajo Práctico

Para organizar el desarrollo de este trabajo práctico de manera efectiva, recomendamos modularizar las diferentes funcionalidades en archivos .py y carpetas separadas, facilitando así la reutilización de código y la depuración. Una posible estructura de entrega podría ser:

```
Apellido_Nombre_TP4.zip
|- data/
    |- raw/           # Datos originales sin modificar
        |- toyota_dev.csv
        |- toyota_test.csv
    |- processed/     # Datos procesados y curados

|- src/
    |- utils/        # Funciones auxiliares (ej: seteo de logs)
    |- models/       # Modelos de ML implementados
    |- callbacks/    # Funciones que manejan eventos (ej: Early Stopping)
    |- schedulers/   # Funciones para ajustar dinámicamente el learn rate
    |- train&test/   # Scripts para entrenar y evaluar los modelos
    |- datasets/     # Scripts para cargar y preprocesar datasets
    |- dataloaders/  # Scripts para gestionar el acceso a los datos

|- logs/           # Logs de ejecución, métricas y errores

|- results/        # Resultados obtenidos del entrenamiento y pruebas

|- runs/           # Generadas por los experimentos (ej: checkpoints)

|- configs/        # Configuración con hiperparámetros y ajustes

|- notebooks/
    |- Entrega_TP4.ipynb    # Respuestas de todos los ejercicios del TP

|- main.py    # Script para ejecutar el pipeline completo del experimento

|- go.sh      # Script para ejecutar el experimento desde la terminal

|- requirements.txt       # Especificar dependencias del proyecto

|- README.md              # Descripción del TP e instrucciones de uso
```

Esta estructura es opcional y flexible. Se pueden agregar o eliminar archivos y/o carpetas según sea necesario, pero es obligatorio incluir un Jupyter Notebook con todas las respuestas a los ejercicios.