Actividad Práctica

Minería de datos: Aspectos Avanzados

20 de febrero, 2025

Regresión/Clasificación ordinal (0,5 ptos)

Objetivo:
Implementar la idea básica del método propuesto por Cardoso y Sousa en su artículo *"Measuring the Performance of Ordinal Classification"*, utilizando Python. La implementación debe centrarse en la técnica principal descrita, sin necesidad de profundizar en todos los aspectos teóricos o extensiones avanzadas. En otras palabras, bastaría con implementar la metodología descrita en la **Sección 2.1** para casos lineales.

https://www.jmlr.org/papers/volume8/cardoso07a/cardoso07a.pdf

Descripción de la tarea:

1. **Implementación:** Desarrollar un código en Python que implemente la técnica base para la regresión ordinal según la publicación de referencia. Se recomienda utilizar bibliotecas como NumPy y scikit-learn para facilitar el desarrollo. Igualmente, no se prohíbe el uso de LLMs para ayudar a la tarea de codificación.
2. **Experimentación:** Evaluar la implementación en los cuatro conjuntos de datos estándar comúnmente utilizados para problemas de regresión ordinal:
    - **LEV:** *Level of Emotional Valence*
    - **ERA:** *Employee Retention Analysis*
    - **ESL:** *Elementary School Level*
    - **SWD:** *Software Defect Prediction*
3. **Evaluación:** Analizar el rendimiento del método mediante métricas apropiadas para problemas de regresión ordinal, como el *Mean Absolute Error (MAE)*. Para ello, hay que usar al menos dos clasificadores base distintos.

Entregable:

1. Notebook en Python, bien documentado y organizado.
2. Informe breve (incluido en el mismo notebook) explicando:
    - El enfoque de implementación.
    - Los resultados experimentales en cada dataset.
    - Un análisis crítico de los resultados y posibles limitaciones.

Aprendizaje Semi-supervisado (0,5 ptos)

Objetivo:
Implementar la técnica de *Co-Training* propuesta por Blum y Mitchell (1998) en Python, utilizando dos clasificadores base y aplicando la técnica sobre un conjunto de datos adecuado para el paradigma de aprendizaje semi-supervisado.

Descripción de la tarea:

1. **Implementación:** Desarrollar un código en Python que implemente el algoritmo de Co-Training. La implementación debe incluir:
    - Dos clasificadores base, preferiblemente modelos clásicos como *Logistic Regression*, *Decision Tree* o *SVM*, disponibles en *scikit-learn*.
    - División del espacio de características del dataset en dos *vistas*, generadas mediante la partición disjunta y balanceada de las características en dos subconjuntos de igual tamaño.
    - Proceso iterativo de Co-Training: cada clasificador entrena con su respectiva vista y etiqueta ejemplos no supervisados para el otro clasificador.
2. **Recomendaciones:**
   Se recomienda utilizar el conjunto de datos *"Covertype"* del UCI Machine Learning Repository. Este dataset tiene múltiples características que permiten una partición equilibrada en dos vistas. Alternativamente, se puede utilizar el dataset *"Digits"* de *scikit-learn*, simplificado para tareas de clasificación binaria. No obstante, el estudiante podrá elegir el data set que prefiera.

   No se prohíbe el uso de LLMs para ayudar a la tarea de codificación. Existen implementaciones de este algoritmo en algunos repositorios de Github, pero datan de más de 7 años, por lo que tendrán problemas de compatibilidad. Los estudiantes pueden utilizar los recursos que quieran.
3. **Experimentación:**
    - Dividir el dataset en un conjunto pequeño de datos etiquetados y un conjunto grande de datos no etiquetados.
    - Entrenar los clasificadores base inicialmente con los datos etiquetados.
    - Aplicar Co-Training para etiquetar progresivamente el conjunto no supervisado.
    - Evaluar el rendimiento final utilizando métricas como *Accuracy* y *F1-score*.

Auto Machine Learning (0,5 ptos)

Objetivo:
El objetivo de este ejercicio es comparar el rendimiento de diferentes herramientas de AutoML para la clasificación de datos tabulares. Usaremos el conjunto de datos *Wine Dataset* de `sklearn` y evaluaremos las predicciones en términos de precisión (*accuracy*), F1-score y tiempo de entrenamiento.

Tareas a realizar:

1. **Preparación de datos:**
   - Cargar el *Wine Dataset* de `sklearn`.
   - Dividir en conjunto de entrenamiento (70%) y prueba (30%).
   - Estandarizar las características usando `StandardScaler`.
2. **Entrenamiento y evaluación de modelos:**
   Entrenar y comparar los siguientes modelos:
   - **XGBoost:** Utilizar `XGBClassifier` con parámetros por defecto.
   - **TabPFN:** Usar el clasificador `TabPFNClassifier` con 32 configuraciones de ensemble (`N_ensemble_configurations=32`).
   - **Auto-sklearn:** Configurar `AutoSklearnClassifier` con un tiempo máximo de 5 minutos (`time_left_for_this_task=300`) y límite por modelo de 1 minuto (`per_run_time_limit=60`).
   - **[OPTATIVO] H2O AutoML:** Entrenar con un tiempo máximo de 5 minutos (`max_runtime_secs=300`).
3. **Evaluación:**
   Comparar los modelos según:
   - *Accuracy*: Precisión global de la clasificación.
   - *F1 Score*: Promedio ponderado entre precisión y exhaustividad.
   - *Tiempo de entrenamiento*: Duración total del ajuste del modelo.
   - *Tiempo de predicción*: Duración para predecir el conjunto de prueba.
4. **Análisis y conclusiones:**
   - Analizar cuál modelo ofrece el mejor equilibrio entre rendimiento y tiempo.
   - Comentar si la simplicidad y velocidad de TabPFN compensa frente a las soluciones de AutoML más tradicionales.

Entregable:

1. Código Python bien documentado en un solo script (notebook `.ipynb`).

2. Tabla comparativa de resultados y breve informe (dentro del notebook) con las conclusiones obtenidas.

La entrega se efectuará subiendo la documentación a la actividad correspondiente de PRADO, antes del ***20 de Marzo de 2025 (23:55)***.