Examen de Optimización - Unidad II

Universidad Nacional del Altiplano - FINESI
Apellidos y Nombres: Mamani Huatta Cliver Daniel
Código: 231211

1. ¿Cuáles son las principales diferencias entre la optimización de funciones convexas y no convexas, y cómo influyen en la elección de métodos?

Respuesta:

Las funciones convexas cualquier punto mínimo que encontremos será el mínimo global de toda la función. Esto significa que los algoritmos de optimización siempre encontrarán la mejor solución posible. Además, estos algoritmos convergen de manera más rápida y predecible. Las funciones no convexas presentan múltiples mínimos locales, lo que puede hacer que los algoritmos se "queden atrapados.[en] soluciones que no son las mejores globalmente.

2. ¿Cuándo resulta más efectivo usar variantes del descenso de gradiente y cuáles son los factores clave en su selección?

Respuesta:

El descenso de gradiente clásico es efectivo para datasets pequeños y funciones suaves. SGD es mejor para datasets grandes por sus actualizaciones frecuentes. Mini-batch equilibra eficiencia y estabilidad. Las variantes adaptativas (Adam, RMSprop) son útiles cuando hay características con diferentes escalas.

Factores clave: tamaño del dataset, recursos computacionales, naturaleza del problema y velocidad de convergencia requerida.

3. ¿Cómo se aplican los criterios de optimalidad en un método de optimización?

Respuesta:

Los criterios de optimalidad verifican si hemos encontrado una solución óptima:

- Primer orden: El criterio de primer orden establece que en el punto óptimo, el gradiente debe ser cero (o muy pequeño en la práctica) y se verifica calculando $\nabla f(x^*) = 0$ (gradiente nulo)
- Segundo orden: Examina la curvatura de la función usando el Hessiano. Si el Hessiano es positivo definido, confirmamos que tenemos un mínimo local.
- KKT: Karush-Kuhn-Tucker para problemas con restricciones, que incluyen condiciones de factibilidad y complementariedad.

En la práctica, se implementan con tolerancias numéricas apropiadas.

4. ¿De qué manera facilitan las herramientas de optimización automática (p.ej., Optuna) la búsqueda de configuraciones óptimas en aprendizaje automático?

Respuesta:

Optuna automatiza la búsqueda de hiperparámetros usando algoritmos inteligentes como TPE que aprenden de resultados anteriores. Incluye "pruning"para detener experimentos no prometedores y permite paralelización. Esto reduce significativamente el tiempo de experimentación y mejora los resultados con menos recursos computacionales.

5. ¿Cómo influye la selección de estrategias de regularización en el desempeño y la generalización de un modelo?

Respuesta:

La regularización ayuda a prevenir el sobreajuste controlando la complejidad del modelo. Ridge (L2) suaviza los pesos del modelo, mientras que Lasso (L1) puede eliminar características irrelevantes automáticamente. Elastic Net combina ambos beneficios.

Mejora la generalización reduciendo la varianza, haciendo el modelo más estable y menos sensible al ruido, resultando en mejor rendimiento en datos nuevos.

6. ¿Cómo se pueden integrar rutinas de validación y corrección de código en Optuna para mejorar la calidad y eficacia del proceso de optimización de hiperparámetros?

Respuesta:

Se integran definiendo rangos válidos para parámetros y manejando excepciones:

```
def funcion_objetivo(trial):
    # Validacion de parametros
    tasa_aprendizaje = trial.suggest_float('tasa_aprendizaje', 1e-5, 1e
        -1)
    assert tasa_aprendizaje > 0, "La tasa de aprendizaje debe ser
        positiva"

     # Validacion de configuracion
     modelo = crear_modelo(trial)
     return entrenar_y_evaluar(modelo)
```

Esto mejora la calidad detectando errores tempranamente y evitando configuraciones inválidas.

7. ¿Cómo podemos utilizar las métricas provistas por scikit-learn para evaluar un modelo de clasificación, y qué información adicional brindan además de la exactitud?

Respuesta:

```
from sklearn.metrics import accuracy_score, precision_score,
    recall_score, f1_score, classification_report, confusion_matrix
y_true = [0, 1, 2, 2, 2]
y_pred = [0, 2, 2, 2, 1]
print("Accuracy:", accuracy_score(y_true, y_pred))
print("Precision:", precision_score(y_true, y_pred, average='macro'))
print("Recall:", recall_score(y_true, y_pred, average='macro'))
print("F1-score:", f1_score(y_true, y_pred, average='macro'))
print("Classification Report:")
print(classification_report(y_true, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_true, y_pred))
```

- **Precision**: Proporción de predicciones positivas correctas

- **Recall**: Proporción de casos positivos identificados
- **F1-score**: Balance entre precisión y recall
- **Matriz de confusión**: Muestra qué clases se confunden
- **Classification report**: Resumen completo por clase

Estas métricas son especialmente útiles para datasets desbalanceados donde la exactitud puede ser engañosa.