Examen Resuelto Optimización

Etzel Yuliza Peralta López

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

Optimización convexa: Una función convexa tiene la propiedad de que cualquier línea recta entre dos puntos de su gráfica está por encima o en la misma posición que la gráfica de la función. Esto implica que todo mínimo local también es un mínimo global, lo que facilita la optimización.

Ejemplos comunes de funciones convexas incluyen la regresión lineal con regularización L2 (Ridge), máquinas de soporte vectorial lineales (SVM lineales) y programación lineal o cuadrática.

Optimización no convexa: Las funciones no convexas pueden presentar múltiples mínimos locales y máximos, lo que complica la identificación del mínimo global. Esto es típico en redes neuronales profundas, clustering no lineal, y modelos complejos de portafolios financieros.

Influencia en la elección de métodos:

- Para funciones convexas, se prefieren métodos con garantías teóricas de convergencia al óptimo global, como el descenso de gradiente estándar, métodos de Newton, y métodos cuasi-newtonianos como BFGS.
- Para funciones no convexas, se recurre a métodos más robustos frente a múltiples óptimos locales, como el descenso de gradiente estocástico (SGD), Adam, técnicas heurísticas (p. ej., algoritmos genéticos), y métodos bayesianos. La aleatoriedad y la inicialización cobran gran importancia, por lo que se suelen aplicar técnicas de rearranque aleatorio o enfoques multi-inicio.

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

Cuándo usar variantes:

- Con grandes volúmenes de datos, es más eficiente usar el descenso de gradiente estocástico (SGD) que procesa una muestra por iteración, reduciendo el costo computacional.

- Cuando el entrenamiento es inestable o muy lento, se recomiendan variantes como Momentum, RMSProp, Adam o Adagrad para mejorar la convergencia.

Factores clave en la selección:

- *Tamaño del dataset:* Para datasets grandes, SGD o mini-batch son recomendables; para datasets pequeños, puede usarse descenso batch completo.
- *Topología del problema:* Para problemas no convexos, métodos adaptativos como Adam o RMSProp son más eficaces.
- *Estabilidad de gradientes:* Si los gradientes varían ampliamente, Adam y RMSProp ayudan a estabilizar el entrenamiento.
- *Requerimientos de memoria:* Algunos métodos (como Adam) requieren más memoria, lo cual debe considerarse en dispositivos con limitaciones.
- *Presencia de ruido:* SGD con Momentum o Nesterov suaviza las actualizaciones frente a ruido en los datos.
- *Reproducibilidad:* El gradiente batch es más reproducible al ser determinista, mientras que los métodos estocásticos pueden requerir fijar semillas aleatorias.

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

Los criterios de optimalidad permiten identificar si un punto es solución óptima. Existen dos tipos principales:

Criterios de primer orden (gradiente): Si $\nabla f(x^*) = 0$, entonces x^* es un punto crítico. En funciones convexas, esto implica un mínimo global. En funciones no convexas, puede ser mínimo local, máximo local o punto de silla.

Criterios de segundo orden (Hessiano): Se evalúa la matriz Hessiana $H(x) = \nabla^2 f(x)$. Si:

- $H(x^*)$ es definida positiva $\Rightarrow$ mínimo local.
- $H(x^*)$ es definida negativa $\Rightarrow$ máximo local.
- $H(x^*)$ es indefinida $\Rightarrow$ punto de silla.

Aplicación práctica: Los algoritmos de optimización iterativa suelen usar estos criterios para detener el proceso:

- $\|\nabla f(x_k)\| < \varepsilon$
- Cambio insignificante en la función objetivo.
- Límite de iteraciones alcanzado.

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

Las herramientas de optimización automática ayudan a encontrar hiperparámetros óptimos para modelos de machine learning de manera eficiente, inteligente y reproducible. Aunque Optuna es un ejemplo destacado, este tipo de herramientas en general comparten funcionalidades clave.

Ventajas generales:

- **Automatización del proceso:** Permiten definir el espacio de búsqueda, la métrica objetivo, y gestionan las ejecuciones sin intervención humana.
- **Exploración eficiente:** Utilizan algoritmos avanzados (como TPE, búsqueda bayesiana, algoritmos evolutivos) para explorar el espacio.
- **Técnicas de poda y early stopping:** Interrumpen configuraciones poco prometedoras para ahorrar tiempo.
- **Integración con frameworks populares:** Se integran fácilmente con Scikit-learn, XGBoost, PyTorch, LightGBM, entre otros.
- **Repetibilidad y visualización:** Guardan resultados y permiten analizar visualmente la importancia de cada hiperparámetro.

Ejemplo de código con Optuna:

```
import optuna

def objective(trial):
C = trial.suggest_loguniform("C", 1e-3, 1e2)
gamma = trial.suggest_loguniform("gamma", 1e-4, 1e1)
clf = SVC(C=C, gamma=gamma)
return cross_val_score(clf, X, y, cv=3).mean()

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=100)
```

Otras herramientas similares: Hyperopt, Ray Tune, Ax, Keras Tuner, entre otras. Todas comparten el objetivo de optimizar configuraciones complejas de forma automática y escalable.

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

La regularización es esencial para controlar el sobreajuste en modelos de machine learning, mejorando la capacidad del modelo para generalizar a nuevos datos.

Impacto en el desempeño:

- Durante el entrenamiento, la regularización puede disminuir ligeramente la precisión al imponer restricciones al modelo.
- En validación y prueba, mejora el rendimiento al prevenir que el modelo memorice los datos específicos del entrenamiento.

Tipos comunes de regularización:

- **L1 (Lasso):** Penaliza la suma de los valores absolutos de los coeficientes. Promueve la sparsidad y permite seleccionar características relevantes.
- **L2 (Ridge):** Penaliza la suma de los cuadrados de los coeficientes. Disminuye la magnitud sin eliminar variables.
- **ElasticNet:** Combinación de L1 y L2, busca un balance entre selección de características y estabilidad del modelo.
- **Dropout:** Apaga aleatoriamente neuronas durante el entrenamiento, reduciendo co-adaptaciones.
- **Early stopping:** Detiene el entrenamiento si no hay mejora significativa en el conjunto de validación tras cierto número de épocas.

Influencia en la generalización:

- Reduce la complejidad del modelo, limitando su capacidad para memorizar ruidos del conjunto de entrenamiento.
- Obliga al modelo a aprender patrones robustos que se generalicen mejor a nuevos datos.
- Mejora el rendimiento en escenarios con datos limitados o ruidosos, actuando como una forma de control estructural del modelo.

Selección adecuada: Depende de la naturaleza de los datos y del modelo. Por ejemplo:

- L1 es útil si se sospecha que muchas variables son irrelevantes y se desea realizar selección de características.
- L2 es apropiado cuando se busca mantener todas las variables pero evitar coeficientes grandes.
- ElasticNet es ideal cuando se desea combinar ambos beneficios.
- Dropout es preferido en redes neuronales profundas para mitigar la sobredependencia entre neuronas.
- Early stopping es especialmente eficaz en modelos con alto número de iteraciones o épocas.

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?

Integrar validación y control de errores es clave para que las herramientas de optimización automática (como Optuna) funcionen de manera robusta y eficiente. Estas estrategias también son aplicables en otros entornos automatizados.

Técnicas recomendadas:

- **Validación cruzada (Cross-Validation):** Usar K-Fold dentro de la función objetivo permite evaluar el rendimiento promedio y reducir el riesgo de sobreajuste a una sola partición.
- **Control de errores (try-except):** Previene que errores de ejecución detengan todo el estudio. Por ejemplo:

```
try:
model.fit(X_train, y_train)
except Exception:
return float('inf')
```

- **Restricciones lógicas y validaciones internas:** Verificar que los hiperparámetros propuestos estén dentro de rangos válidos y tengan sentido lógico según el modelo (por ejemplo, evitar que el número de árboles sea negativo o que el learning rate sea cero).
- **Callbacks y monitoreo:** Usar funciones de callback para guardar modelos si superan cierto umbral de desempeño, enviar alertas o actualizar dashboards en tiempo real.
- **Pruning y logging:** La poda permite detener ensayos que no muestran progreso. El logging permite rastrear la evolución del entrenamiento y depurar si es necesario.
- **Verificación de datos:** Antes del entrenamiento, revisar escalado, ausencia de NaN, tipo de datos correcto, codificación de variables categóricas y balance de clases.

Beneficios:

- Mejora la estabilidad del proceso de optimización al reducir errores inesperados.
- Evita configuraciones inválidas o fallos catastróficos.
- Asegura que las métricas reportadas reflejan un comportamiento realista y reproducible del modelo.
- Facilita la depuración y documentación de los experimentos para replicabilidad y análisis posterior.

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?

Scikit-learn ofrece múltiples métricas para evaluar modelos de clasificación más allá de la exactitud (accuracy). Estas métricas permiten comprender mejor el rendimiento del modelo, especialmente en casos de clases desbalanceadas.

Código de ejemplo:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score
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'))
```

Explicación de las métricas:

- **Accuracy:** Proporción de predicciones correctas sobre el total. Puede ser engañosa si hay clases desbalanceadas.
- **Precision:** Proporción de verdaderos positivos entre todas las predicciones positivas. Con macro promedio, se calcula la media de precisión para cada clase, útil cuando todas las clases son igual de importantes.
- **Recall (Sensibilidad):** Proporción de verdaderos positivos entre todos los casos reales positivos. Con macro promedio, se evalúa la capacidad del modelo para encontrar todos los ejemplos de cada clase.

Métricas adicionales útiles:

- **F1-score:** Media armónica entre precisión y recall.
- **Matriz de confusión:** Muestra visualmente los aciertos y errores por clase.
- **ROC AUC:** Para clasificadores binarios, mide el área bajo la curva ROC.

Estas métricas ofrecen una perspectiva más completa del desempeño del modelo y son fundamentales para mejorar y comparar modelos de clasificación en contextos reales.